- **企业级开发** 从产品经理和设计师的角度深入浅出地介绍Android App从开发、调试到上线的企业级开发流程
- **范例丰富实用** 提供3个主流App与11个趣味开发范例，每个范例均给出设计思路与示例代码
- **技术先进，涉及面广** 包括卫星导航、Socket通信、多点触控、百叶窗动画、蓝牙技术、支付SDK、三端融合等众多热点技术





—— Android Studio 2.X/Android 7.X企业级开发 ——

Android Studio 开发实战

从零基础到App上线

欧阳燊 编著

清华大学出版社

移动开发丛书



Android Studio开发实战
从零基础到App上线

欧阳燊 编著

清華大学出版社
北京

内 容 简 介

本书是一部Android开发的实战教程，由浅入深、由基础到高级，带领读者一步一步走进App开发的神奇世界。

全书共分为16章。其中，前8章是基础部分，主要讲解Android Studio的环境搭建、App开发的各种常用控件、App的数据存储方式、如何调试App并将App发布上线；后8章是进阶部分，主要讲解App开发的设备操作、网络通信、事件、动画、多媒体、融合技术、第三方开发包、性能优化等。书中在讲解知识点的同时给出了大量实战范例，方便读者迅速将所学的知识运用到实际开发中。通过本书的学习，读者能够掌握3类主流App的基本开发技术，包括购物App（电子商务）、聊天App（即时通信）、打车App（交通出行）。另外，能够学会开发一些趣味应用，包括简单计算器、房贷计算器、万年历、日程表、手机安全助手、指南针、卫星浑天仪、抠图工具、动感影集、影视播放器、音乐播放器、WIFI共享器等。

本书适用于Android开发的广大从业者、有志于转型App开发的程序员、App开发的业余爱好者，也可作为大中专院校与培训机构的Android课程教材。

图书在版编目（CIP）数据

Android Studio开发实战：从零基础到App上线/欧阳燊编著. —北京：清华大学出版社，2017
（移动开发丛书）
ISBN 978-7-302-47006-9

Ⅰ. ①A… Ⅱ. ①欧… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆CIP数据核字（2016）第100219号

责任编辑： 王金柱
封面设计： 王　翔
责任校对： 闫秀华
责任印制：

出版发行： 清华大学出版社
　　网　　址： http://www.tup.com.cn，http://www.wqbook.com
　　地　　址： 北京清华大学学研大厦A座　　**邮　　编：** 100084
　　社 总 机： 010-62770175　　**邮　　购：** 010-62786544
　　投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
　　质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn

印 装 者：
经　　销： 全国新华书店
开　　本： 190mm×260mm　**印　张：** 41.5　**插　页：** 2　**字　　数：** 1062千字
版　　次： 2017年6月第1版　　**印　　次：** 2017年6月第1次印刷
印　　数： 3001~6000
定　　价： 128.00元

产品编号：072654-01

推 荐 序

计算机的发展是以信息智能化与小型化为进化路线，从 IBM 庞大的巨型机到比尔盖茨的个人电脑，信息无所不在。乔布斯的伟大之处在于“用一个手指头改变世界”。当全世界的粉丝用苹果手机的时候，移动开发领域开始全面地封闭在 iOS 的体系里。安卓作为移动手机和设备开放象征的另一级，更具有活力和前途。

欧阳先生是一位具有丰富程序开发经验的架构师和项目管理者，平时常常思考和总结 21 世纪以来我国软件开发者，特别是移动开发开发工程师的困惑。社会从“一支笔的科学家时代”发展到“一个键盘开发 App 改变世界”，对程序员来说，用自己的智慧进行移动应用开发是创业的捷径。读者遵循书中的指引，很快能够登堂入室，成为当前安卓应用开发的精英人才。

本书对所有有志于进行安卓系统开发的人员而言具有非常重要的意义。

杭州海适云承科技有限公司

董事长兼首席架构师

沈英桓

前　　言

移动应用开发又称 App 开发，是近年来的新兴软件开发行业。基于手机设备的特性，App 开发与服务器开发、网页开发等传统软件开发有很大不同，将 App 开发相关技术称为一门新兴学科也不为过。

作为一门学科，必然要求建立一套理论体系，这个理论体系应当具有普遍性与适用性，不会随着工具的变迁而消亡。App 开发就是如此，无论使用 Android 开发还是 iOS 开发，所采用的技术、要实现的功能都大同小异，区别在于需要使用不同的编程工具进行开发。对于用户来说，华为手机上的微信与苹果手机上的微信都是社交 App，这两个微信在功能和使用上并没有显著区别。

笔者从事软件开发工作十几年，期间经历了多次编程方向的转型，先从 C/C++开发转向 Java 开发，再从 Java 开发转向 Android 开发，而 Android 开发先用 ADT 后用 Android Studio。在多次转型过程中，笔者深深体会到，无论是编程语言还是开发工具，变化的都是技术实现手段，而不是人类愿景和系统原理。人类愿景是让生活更加便捷、让娱乐更加丰富，系统原理是让软件界面更加美观、让运行速度更加流畅。

本书的写作目的是教会读者 Android 开发，带领读者走进一个崭新的学科领域。市面上的 Android 开发书籍林林总总，写作风格各有千秋，不过讲解的基本是编程开发，有的还会讲解项目管理。本书除了介绍常规的 Android 开发外，还尝试从两方面加以拓展，一方面从产品经理的角度仔细分析 App 技术能帮用户做什么事情、能带给用户什么收获；另一方面从设计师的角度详细论述如何把千篇一律的页面变得生动活泼，如何让某个功能实现得更合理、高效。

全书的内容编排采用由浅入深、循序渐进的章节体例，不但考虑初学者的学习连续性，而且可以建立一个统一、连贯的学科体系。这么编排的好处是显而易见的，读者只要按照顺序学习，就能在学习过程中对已学部分不断复习巩固，同时提前预习后面的技术点，一方面衔接自然，另一方面提高学习效率。比如第 3 章末尾介绍实战项目“登录 App”，紧接着第 4 章开头介绍如何实现登录页面的记住密码功能；第 12 章介绍“动画”，一方面为前一章的飞掠横幅补充动画效果，另一方面为后一章的相册切换动画埋下伏笔。

全书可分为两大部分，第一部分是第 1～8 章，主要介绍 Android Studio 的环境搭建，App 开发的各种常用控件，App 的数据存储方式。如何调试 App 并将 App 发布上线，这部分囊括了 App 开发的基础知识，特别详细说明 App 从开发到调试再到上线的企业级开发流程。第二

部分是第9～16章，主要介绍App开发的高级部分，包括设备操作、网络通信、事件、动画、多媒体、融合技术、第三方开发包、性能优化等，这部分涵盖App开发的进阶内容，与第一部分相比就像是“鸟枪换炮”，让开发者完成从游击队到正规军的华丽转变。

建议初学者和在校学生完整学习第1～8章内容，因为这部分包含App开发的必备技能，只有打好基础，才能进一步学习。至于第9～16章内容，根据前面的学习情况和个人兴趣爱好选择相应的章节学习即可。如果倾向于学习工具类App的开发，就可以选择学习“第9章 设备操作”“第11章 事件”“第12章 动画”“第13章 多媒体”；如果倾向于学习企业类App的开发，就可以选择学习“第10章 网络通信”“第14章 融合技术”“第15章 第三方开发包”“第16章 性能优化”。

对于有经验的开发者来说，可以自行选择不熟悉的知识点拾遗补缺。另外，本书讲述的部分知识点很具特色，如卫星导航、Socket通信、多点触控、百叶窗动画、音乐播放器、蓝牙技术、支付SDK、图片缓存原理等，这些内容在同类Android入门书籍中鲜有论述，有兴趣的读者可重点关注。

当然，本书面向的读者不仅是开发人员和计算机专业学生，也包括移动互联网行业的其他从业人员。对于产品经理来说，可以了解一下某个功能使用的技术，看似简单的功能，也许并不容易实现。对于设计师来说，“他山之石，可以攻玉”，可以参考一下别人的实现方式，也许正好可以激发你的灵感，其实不无裨益。对于测试人员来说，可以熟悉一下每项技术的优缺点，从而制订出更全面的测试方案，也许能发现更多BUG。

本书所有代码都基于Android Studio 2.2.3开发，并使用API 25的SDK（Android 7.1.1）编译与调试通过。读者在阅读本书时，若对书中内容有疑问，可在笔者的博客（http://blog.csdn.net/aqi00）留言。

本书范例的素材和代码下载地址为：http://pan.baidu.com/s/1dFEFEhF（注意区分数字和英文字母大小写）。如果下载有问题，请发送电子邮件至booksaga@126.com，邮件主题设置为“求从零基础到App上线下载资源”。

最后，感谢王金柱编辑的热情指点，感谢我的家人一直以来的支持，没有他们的鼎力相助，本书就无法顺利完成。

欧阳燊

2017年1月

目　　录

第1章　Android Studio 环境搭建……1
1.1　Android Studio 简介……2
1.2　Android Studio 的安装……2
1.2.1　开发机配置要求……2
1.2.2　安装依赖的软件……3
1.2.3　安装 Android Studio……5
1.3　运行小应用 Hello World……7
1.3.1　创建新项目……7
1.3.2　编译项目/模块……10
1.3.3　创建模拟器……10
1.3.4　在模拟器上运行 App……11
1.4　App 的工程结构……12
1.4.1　工程目录说明……12
1.4.2　编译配置文件 build.gradle……13
1.4.3　App 运行配置 AndroidManifest.xml……15
1.4.4　在代码中操纵控件……15
1.5　准备开始……17
1.5.1　使用快捷键……17
1.5.2　安装 SVN 工具……18
1.5.3　安装常用插件……19
1.5.4　导入 ADT 工程……21
1.6　小结……22
第2章　初级控件……23
2.1　屏幕显示……24
2.1.1　像素……24
2.1.2　颜色……25
2.1.3　屏幕分辨率……26
2.2　简单布局……27
2.2.1　视图 View 的基本属性……27
2.2.2　线性布局 LinearLayout……30
2.2.3　滚动视图 ScrollView……32
2.3　简单控件……34
2.3.1　文本视图 TextView……34
2.3.2　按钮 Button……38
2.3.3　图像视图 ImageView……39
2.3.4　图像按钮 ImageButton……43
2.4　图形基础……45
2.4.1　图形 Drawable……46
2.4.2　状态列表图形……47

2.4.3 形状图形 48
2.4.4 九宫格图片 51
2.5 实战项目：简单计算器 52
2.5.1 设计思路 53
2.5.2 小知识：日志 Log/提示 Toast 54
2.5.3 代码示例 55
2.6 小结 58
第 3 章 中级控件 59
3.1 其他布局 60
3.1.1 相对布局 RelativeLayout 60
3.1.2 框架布局 FrameLayout 64
3.2 特殊按钮 65
3.2.1 复选框 CheckBox 65
3.2.2 开关按钮 Switch 66
3.2.3 单选按钮 RadioButton 67
3.3 适配视图基础 68
3.3.1 下拉框 Spinner 68
3.3.2 数组适配器 ArrayAdapter 69
3.3.3 简单适配器 SimpleAdapter 70
3.4 编辑框 71
3.4.1 文本编辑框 EditText 72
3.4.2 自动完成编辑框 AutoCompleteTextView 77
3.5 Activity 基础 78
3.5.1 Activity 的生命周期 78
3.5.2 使用 Intent 传递消息 82
3.5.3 向下一个 Activity 传递参数 84
3.5.4 向上一个 Activity 返回参数 85
3.6 实战项目：登录 App 88
3.6.1 设计思路 88
3.6.2 小知识：提醒对话框 AlertDialog 89
3.6.3 代码示例 91
3.7 小结 94
第 4 章 数据存储 95
4.1 共享参数 SharedPreferences 96
4.1.1 基本用法 96
4.1.2 实现记住密码功能 97
4.2 数据库 SQLite 98
4.2.1 SQLite 的基本用法 98
4.2.2 SQLiteOpenHelper 100
4.2.3 优化记住密码功能 106
4.3 SD 卡文件操作 108
4.3.1 SD 卡的基本操作 108
4.3.2 文本文件读写 110
4.3.3 图片文件读写 111

4.4 Application 基础 112
4.4.1 Application 的生命周期 112
4.4.2 利用 Application 操作全局变量 113
4.5 实战项目：购物车 115
4.5.1 设计思路 115
4.5.2 小知识：菜单 Menu 116
4.5.3 代码示例 119
4.6 小结 125
第 5 章 高级控件 126
5.1 日期时间控件 127
5.1.1 日期选择器 DatePicker 127
5.1.2 时间选择器 TimePicker 128
5.2 列表类视图 129
5.2.1 基本适配器 BaseAdapter 129
5.2.2 列表视图 ListView 133
5.2.3 网格视图 GridView 138
5.3 翻页类视图 142
5.3.1 翻页视图 ViewPager 142
5.3.2 翻页标题栏 PagerTitleStrip/PagerTabStrip 145
5.3.3 简单的启动引导页 147
5.4 碎片 Fragment 150
5.4.1 静态注册 150
5.4.2 动态注册/碎片适配器 FragmentStatePagerAdapter 154
5.4.3 改进的启动引导页 157
5.5 Broadcast 基础 159
5.5.1 发送/接收临时广播 159
5.5.2 定时器 AlarmManager 162
5.6 实战项目：日历/日程表 163
5.6.1 设计思路 163
5.6.2 小知识：震动器 Vibrator 165
5.6.3 代码示例 165
5.7 小结 170
第 6 章 自定义控件 171
6.1 自定义视图 172
6.1.1 声明属性 172
6.1.2 构造对象 175
6.1.3 测量尺寸 176
6.1.4 绘制视图 179
6.2 自定义动画 184
6.2.1 任务 Runnable 184
6.2.2 下拉刷新动画 185
6.2.3 圆弧进度动画 186
6.3 自定义对话框 190

6.3.1 对话框 Dialog ······ 190
6.3.2 改进的日期对话框 ······ 191
6.3.3 自定义多级对话框 ······ 195
6.4 自定义通知栏 ······ 195
6.4.1 通知推送 Notification ······ 195
6.4.2 进度条 ProcessBar ······ 198
6.4.3 远程视图 RemoteViews ······ 199
6.5 Service 基础 ······ 202
6.5.1 Service 的生命周期 ······ 203
6.5.2 推送服务到前台 ······ 207
6.6 实战项目：手机安全助手 ······ 210
6.6.1 设计思路 ······ 210
6.6.2 小知识：应用包管理 PackageManager ······ 211
6.6.3 代码示例 ······ 213
6.7 小结 ······ 216
第 7 章 组合控件 ······ 217
7.1 标签栏 ······ 218
7.1.1 标签按钮 ······ 218
7.1.2 实现底部标签栏 ······ 219
7.2 导航栏 ······ 228
7.2.1 工具栏 Toolbar ······ 228
7.2.2 溢出菜单 OverflowMenu ······ 230
7.2.3 搜索框 SearchView ······ 232
7.2.4 标签布局 TabLayout ······ 235
7.3 横幅条 ······ 240
7.3.1 自定义指示器 ······ 241
7.3.2 实现横幅轮播 Banner ······ 243
7.4 增强型列表 ······ 247
7.4.1 循环视图 RecyclerView ······ 247
7.4.2 布局管理器 LayoutManager ······ 252
7.4.3 动态更新循环视图 ······ 256
7.5 实战项目：仿淘宝主页 ······ 258
7.5.1 设计思路 ······ 258
7.5.2 小知识：下拉刷新 SwipeRefreshLayout ······ 259
7.5.3 代码示例 ······ 262
7.6 小结 ······ 266
第 8 章 调试与上线 ······ 267
8.1 调试工作 ······ 268
8.1.1 模拟器调试 ······ 268
8.1.2 真机调试 ······ 272
8.1.3 导出 APK 安装包 ······ 274
8.2 准备上线 ······ 276
8.2.1 版本设置 ······ 276
8.2.2 上线模式 ······ 277



8.2.3 数据加密 281
8.3 安全加固 289
8.3.1 反编译 289
8.3.2 代码混淆 291
8.3.3 第三方加固及重签名 294
8.4 发布到应用商店 296
8.4.1 注册开发者账号 296
8.4.2 创建并提交应用 297
8.5 小结 299
第 9 章 设备操作 300
9.1 摄像头 301
9.1.1 表面视图 SurfaceView 301
9.1.2 使用 Camera 拍照 303
9.1.3 纹理视图 TextureView 308
9.1.4 使用 Camera 2 拍照 309
9.2 麦克风 311
9.2.1 拖动条 SeekBar 312
9.2.2 音量控制 313
9.2.3 录音与播音 314
9.2.4 录像与放映 322
9.3 传感器 326
9.3.1 传感器的种类 327
9.3.2 加速度传感器 328
9.3.3 指南针 330
9.3.4 计步器和感光器 333
9.4 手机定位 334
9.4.1 开启定位功能 334
9.4.2 获取定位信息 337
9.5 实战项目：仿微信的发现功能 341
9.5.1 设计思路 341
9.5.2 小知识：全球卫星导航系统 343
9.5.3 代码示例 345
9.6 小结 351
第 10 章 网络通信 352
10.1 多线程 353
10.1.1 消息传递 Message 353
10.1.2 进度对话框 ProgressDialog 356
10.1.3 异步任务 AsyncTask 359
10.1.4 异步服务 IntentService 365
10.2 HTTP 接口访问 367
10.2.1 网络连接检查 367
10.2.2 移动数据格式 JSON 369
10.2.3 HTTP 接口调用 371
10.2.4 HTTP 图片获取 376

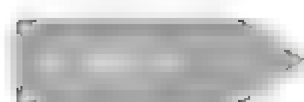
10.3 上传和下载······378
10.3.1 下载管理器 DownloadManager······378
10.3.2 文件对话框······384
10.3.3 文件上传······385
10.4 套接字 Socket······389
10.4.1 网络地址 InetAddress······389
10.4.2 Socket 通信······390
10.5 实战项目：仿手机 QQ 的聊天功能······394
10.5.1 设计思路······394
10.5.2 小知识：可折叠列表视图 ExpandableListView······397
10.5.3 代码示例······401
10.6 小结······407
第 11 章 事件······408
11.1 按键事件······409
11.1.1 检测软键盘······409
11.1.2 检测物理按键······411
11.1.3 音量调节对话框······413
11.2 触摸事件······417
11.2.1 手势事件的分发流程······417
11.2.2 手势事件处理 MotionEvent······421
11.2.3 手写签名······424
11.3 手势检测······426
11.3.1 手势检测器 GestureDetector······426
11.3.2 飞掠视图 ViewFlipper······428
11.3.3 手势控制横幅轮播······431
11.4 手势冲突处理······435
11.4.1 上下滚动与左右滑动的冲突处理······435
11.4.2 内部滑动与翻页滑动的冲突处理······438
11.5 实战项目：抠图神器——美图变变······443
11.5.1 设计思路······443
11.5.2 小知识：图像的基本加工······444
11.5.3 代码示例······445
11.6 小结······451
第 12 章 动画······452
12.1 帧动画······453
12.1.1 帧动画的实现······453
12.1.2 显示 GIF 动画······455
12.1.3 淡入淡出动画······456
12.2 补间动画······457
12.2.1 补间动画的种类······458
12.2.2 补间动画的原理······462
12.2.3 集合动画······465
12.2.4 在飞掠横幅中使用补间动画······466



12.3 属性动画……469
12.3.1 属性动画的用法……469
12.3.2 属性动画组合……472
12.3.3 插值器和估值器……473
12.4 动画的实现手段……477
12.4.1 使用延时重绘……477
12.4.2 设置状态参数……478
12.4.3 滚动器 Scroller……479
12.5 实战项目：仿 QQ 空间的动感影集……481
12.5.1 设计思路……481
12.5.2 小知识：画布的绘图层次……482
12.5.3 代码示例……486
12.6 小结……492
第 13 章 多媒体……493
13.1 相册……494
13.1.1 画廊 Gallery……494
13.1.2 图像切换器 ImageSwitcher……496
13.1.3 图片查看器——青青相册……499
13.2 视频播放……502
13.2.1 视频视图 VideoView……503
13.2.2 媒体控制条 MediaController……505
13.2.3 影视播放器——爱看剧场……507
13.3 内容提供与处理……514
13.3.1 内容提供器 ContentProvider……515
13.3.2 内容解析器 ContentResolver……517
13.3.3 内容观察器 ContentObserver……521
13.4 实战项目：音乐播放器——浪花音乐……524
13.4.1 设计思路……524
13.4.2 小知识：可变字符串 SpannableString……526
13.4.3 代码示例……529
13.5 小结……537
第 14 章 融合技术……538
14.1 网页集成……539
14.1.1 资产管理器 AssetManager……539
14.1.2 网页视图 WebView……540
14.1.3 简单浏览器……542
14.2 JNI 开发……549
14.2.1 NDK 环境搭建……550
14.2.2 创建 JNI 接口……552
14.2.3 JNI 实现加解密……556
14.3 局域网共享……559
14.3.1 无线网络管理器 WifiManager……559
14.3.2 蓝牙 BlueTooth……560



14.4 实战项目：共享经济弄潮儿——WIFI 共享器……570
14.4.1 设计思路……570
14.4.2 小知识：NetBIOS 协议……571
14.4.3 代码示例……574
14.5 小结……582
第 15 章 第三方开发包……583
15.1 地图 SDK……584
15.1.1 查看签名信息……584
15.1.2 百度地图……586
15.1.3 高德地图……591
15.2 分享 SDK……596
15.2.1 QQ 分享……596
15.2.2 微信分享……598
15.3 支付 SDK……602
15.3.1 支付宝支付……603
15.3.2 微信支付……604
15.4 语音 SDK……606
15.4.1 语音识别……606
15.4.2 语音合成……609
15.5 实战项目：仿滴滴打车……610
15.5.1 设计思路……611
15.5.2 小知识：评分条 RatingBar……611
15.5.3 代码示例……614
15.6 小结……615
第 16 章 性能优化……616
16.1 布局文件优化……617
16.1.1 减少重复布局……617
16.1.2 自适应调整布局……619
16.1.3 自定义窗口主题……621
16.2 内存泄漏处理……623
16.2.1 内存泄漏的检测……623
16.2.2 内存泄漏的预防……628
16.3 线程池管理……631
16.3.1 普通线程池……631
16.3.2 定时器线程池……634
16.4 省电模式……634
16.4.1 检测当前电量……635
16.4.2 检测屏幕开关……636
16.5 实战项目：图片缓存框架……638
16.5.1 设计思路……638
16.5.2 小知识：LRU 缓存策略……640
16.5.3 代码示例……642
16.6 小结……649





Android Studio 环境搭建

本章主要介绍如何在个人电脑上安装 Android Studio 和相应的配套环境，并通过一个简单的 App“Hello World”演示 Android Studio 的常用操作与 App 开发、运行的流程，还介绍了 App 的工程结构和开发过程中的准备工作。

1.1 Android Studio 简介

Android 是基于 Linux 的移动设备操作系统，中文名为安卓，主要用于智能手机与平板电脑。Android 与 iOS 为智能手机市场的两大操作系统。在中国大陆，Android 的市场份额更是遥遥领先，据 2016 年 9 月的移动系统调研报告，Android 在中国的市场份额为 85%，其余份额为 iOS。

早期，在 Android 下开发 App 主要使用 Eclipse 和基于 Eclipse 的 ADT。不过 Eclipse 毕竟是为 Java 工程而生的开发平台，并非专门用于 Android，所以先天性不足难以避免。自 2015 年之后，谷歌公司便停止了 ADT 的版本更新，转而重点打造自家的 Android Studio。

Android Studio 是谷歌公司推出的 Android 应用开发环境，与基于 Eclipse 的 ADT 不同，Android Studio 是个全新的开发环境，拥有更强大的功能和更高效的性能。本书使用的 Android Studio 为 2016 年 12 月 6 日发布的 2.2.3 版本，同时支持 Windows、Mac OS X 和 Linux。

使用 Android Studio 比起使用 Eclipse 开发有如下好处：

（1）Android Studio 使用 v7 库与 design 库等只需增加一行配置，而 Eclipse 要想使用这些库得引用整个工程。

（2）高版本的 SDK 与 NDK 只支持 Android Studio，不支持 Eclipse。

（3）更多新功能只能在 Android Studio 中运用，如自动保存、多渠道打包、整合版本管理、支持预览 drawable 文件等。

1.2 Android Studio 的安装

既然 Android Studio 有着众多优点，又是 App 开发大趋势的主流工具，接下来就让我们一步一步地在自己的电脑上安装 Android Studio。

1.2.1 开发机配置要求

工欲善其事，必先利其器。要想保证 Android Studio 的运行速度，开发用的电脑配置就要跟上。现在一般用笔记本电脑开发 App，下面是开发机的基本配置：

（1）内存最低要求 4G，推荐 8G，越大越好。

（2）CPU 要求 1.5GHz 以上，越快越好。

（3）硬盘要求系统盘剩余空间 10G 以上，越大越好。

（4）要求带无线网卡、摄像头，USB 与麦克风正常使用。

（5）如果操作系统是 Windows，那么建议使用 Windows 7 及以上系统版本，因为在 Windows XP 下安装 jdk1.8 时，会提示 Java 8 需要更新版本的 Windows 系统。



1.2.2 安装依赖的软件

Android Studio 作为 Android 应用的开发环境，仍然依赖于 JDK、SDK 和 NDK 三种开发工具。

1. JDK

JDK 是 Java 语言的编译器，全称为 Java Development Kit，即 Java 开发工具包。因为 Android 应用采用 Java 语言开发，所以开发机上要先安装 JDK，下载地址为 http://www.oracle.com/technetwork/java/javase/downloads/index.html。JDK 建议安装 1.8 及以上版本，原因是不同的 Android 版本对 JDK 有相应的要求，如 Android 5.0 默认使用 jdk1.7 编译，Android 7.0 默认使用 jdk1.8 编译。

如果 JDK 为 1.6 或 1.7，而 SDK 为最新版本，就可能导致如下问题：

（1）创建项目后，浏览布局文件设计图时会报错 Android N requires the IDE to be running with Java 1.8 or later。

（2）编译项目失败，提示错误 com/android/dx/command/dexer/Main：Unsupported major.minor version 52.0。

（3）运行 App 失败，提示错误 compileSdkVersion 'android-24' requires JDK 1.8 or later to compile.

装好 JDK 后，还要在环境变量的系统变量中添加 JAVA_HOME，取值为 JDK 的安装目录，例如 D:\Program Files(x86)\Java\jdk1.8.0_102。添加系统变量 CLASSPATH，取值为.;%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\bin。并在系统变量 Path 末尾添加;%JAVA_HOME%\bin。

2. SDK

SDK 是 Android 应用的编译器，全称为 Software Development Kit，即软件开发工具包。SDK 提供了 App 开发的常用工具合集，主要包括：

- build-tools 目录，存放各版本 Android 的各种编译工具。
- docs 目录，存放开发说明文档。
- extras\android 目录，存放兼容低版本的新功能支持库，比如 android-support-v4.jar、v7 的各种库、v13 以上等库。
- platforms 目录，存放各版本 Android 的资源文件。
- platform-tools 目录与 tools 目录，存放常用的开发辅助工具，如数据库管理工具 sqlite3.exe、虚拟机调试监控服务 ddms.bat、九宫格图片制作工具 draw9patch.bat 等。
- samples 目录，存放各版本 Android 常用功能的 demo 源码。
- sources 目录，存放各版本 Android 的 API 开放接口源码。
- system-images 目录，存放模拟器各版本的系统镜像与管理工具。

SDK 可以单独安装，也可以与 Android Studio 一起安装，单独安装的下载页面入口地址是



http://sdk.android-studio.org/。建议与 Android Studio 一起安装，因为这样避免了一些兼容性与环境设置问题。无论是单独安装还是一起安装，装好 SDK 后都要在环境变量的系统变量中添加 ANDROID_HOME，取值为 SDK 的安装目录，例如 D:\Android\sdk。并在系统变量 Path 末尾添加;%ANDROID_HOME%\tools。

SDK 时常有版本更新，可以打开 SDK 安装目录下的 SDK Manager.exe 进行更新操作，该工具的管理窗口如图 1-1 所示。

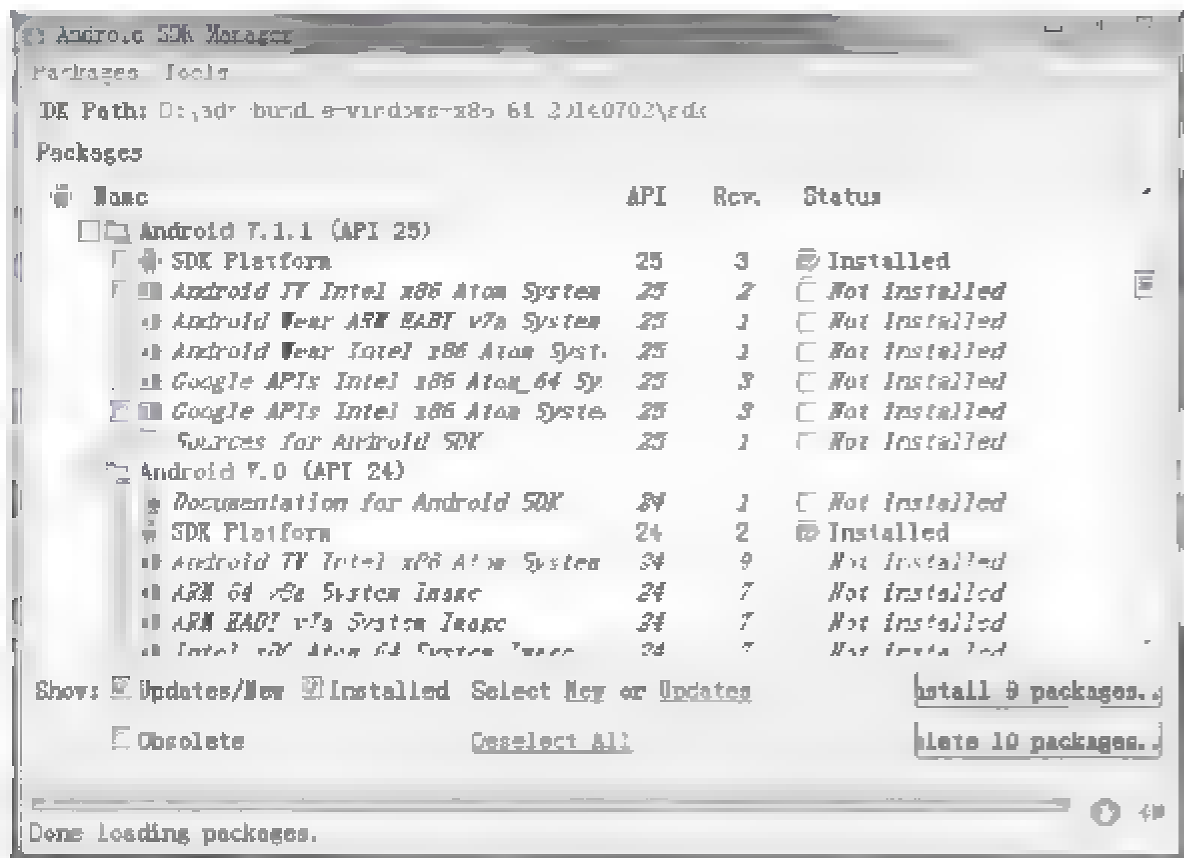


图 1-1　SDK Manager 的管理窗口

首先勾选需要安装或更新的组件，然后单击 Install ** packages 按钮。在下个弹出的窗口页面选中 Accept License，然后单击 Install 按钮，等待安装过程。

如果遇到国外的更新地址无法访问导致安装失败，可采用国内的镜像地址更新，方法是依次选择菜单 Tools→Options，在弹出的设置窗口的 HTTP Proxy Server 栏填写镜像地址的域名，在 HTTP Proxy Port 栏填写镜像地址的端口，然后勾选下面的 Force https://... sources to be fetched using http://...，具体的设置页面如图 1-2 所示。

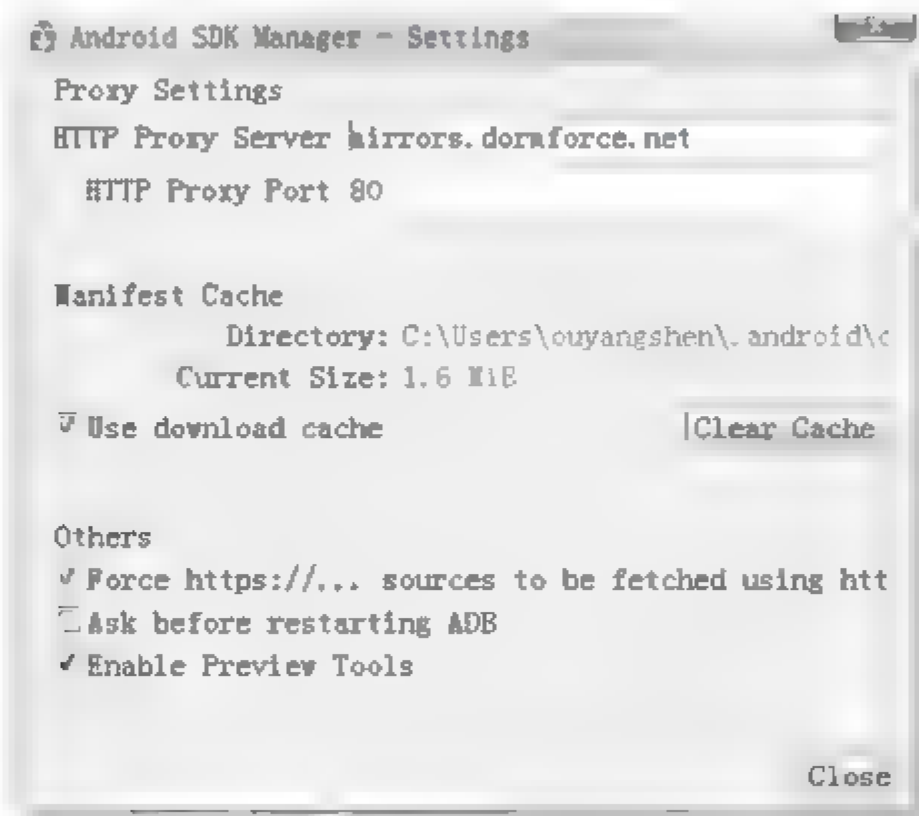


图 1-2　SDK Manager 的设置窗口

下面是设置页面可用的一个国内镜像地址：

腾讯 Bugly，地址：mirrors.dormforce.net，端口：80



3. NDK

NDK 是 C/C++代码的编译器，全称为 Native Development Kit，意即原生开发工具包。该工具包主要供 JNI 接口使用，先把 C/C++代码编译成 so 库，然后由 Java 代码通过 JNI 接口调用 so 库。

NDK 的详细安装步骤见第 14 章的 JNI 部分。装好 NDK 后，要在环境变量的系统变量中添加 NDK_ROOT，取值为 NDK 的安装目录，例如 D:\android-ndk-r12b。然后在系统变量 Path 末尾添加;%NDK_ROOT%。

1.2.3 安装 Android Studio

2016 年 12 月 8 日，谷歌开发者的中文网站上线了。国内开发者可直接在该网站下载 Android Studio，详细的下载页面是 https://developer.android.google.cn/studio/index.html，在这里可以找到 Android Studio 的使用教程。推荐安装带 SDK 的 Android Studio 版本，因为 SDK 内含支持库，自己操作比较费时费力，还容易造成兼容性问题。

双击下载完成的 Android Studio 安装程序，弹出安装界面，如图 1-3 所示。全部勾选安装界面中的选项，然后单击 Next 按钮。在下一页的许可同意页面单击 Agree 按钮，如图 1-4 所示。进入下一页的安装路径配置页面，建议将 Android Studio 和 SDK 装在除系统盘外的其他磁盘（比如 D 盘），然后单击 Next 按钮。

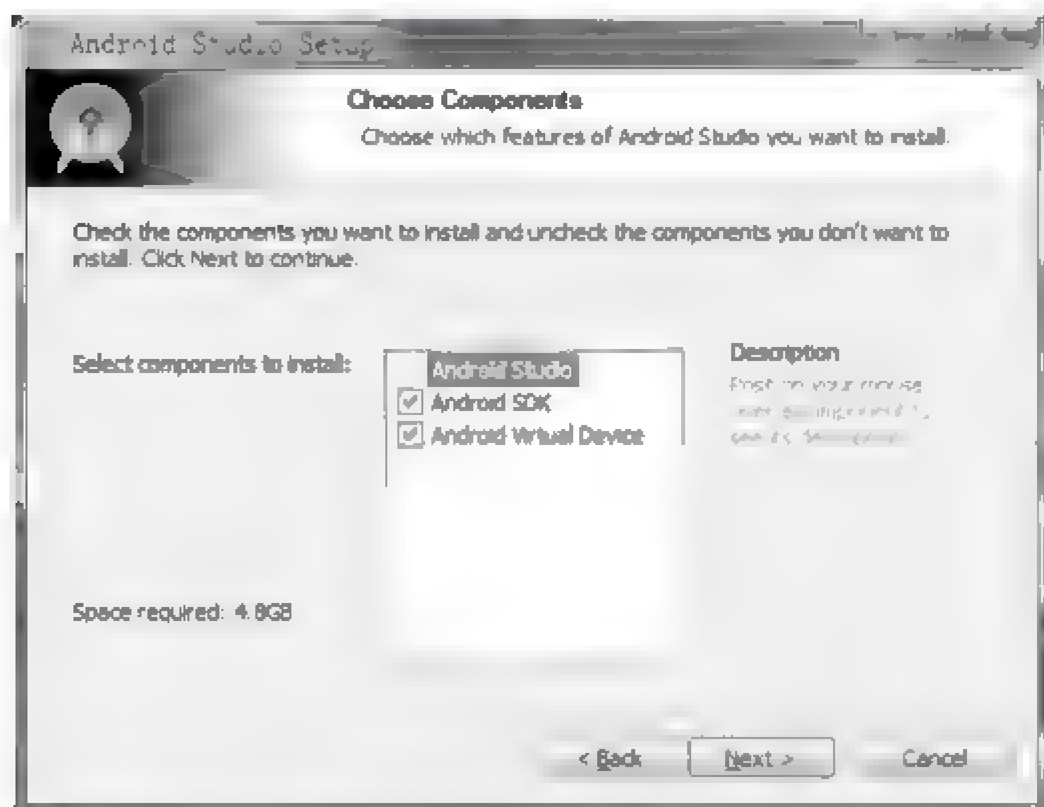


图 1-3 Android Studio 的安装界面



图 1-4 许可同意界面

接下来一直单击 Next 按钮，直到弹出最后一页，单击 Install 按钮，等待安装过程进行。

安装完毕会跳到 Android Studio 的安装向导界面，如图 1-5 所示。单击 Next 按钮进入下一页，如图 1-6 所示。这里保持 Standard 选项，单击 Next 按钮；在配置界面确认 SDK 的安装路径是否正确，确认完毕继续单击 Next 按钮；在最后一个向导界面单击 Finish 按钮，等待设置操作。接下来的下载界面会自动跳转到谷歌网站更新组件，这里直接单击 Cancel 按钮取消下载，然后单击 Finish 按钮结束设置。最后弹出 Welcome to Android Studio 欢迎界面，如图 1-7 所示。单击第一项的 Start a new Android Studio project 即可开始你的 Android 开发之旅。



图 1-5　安装向导一

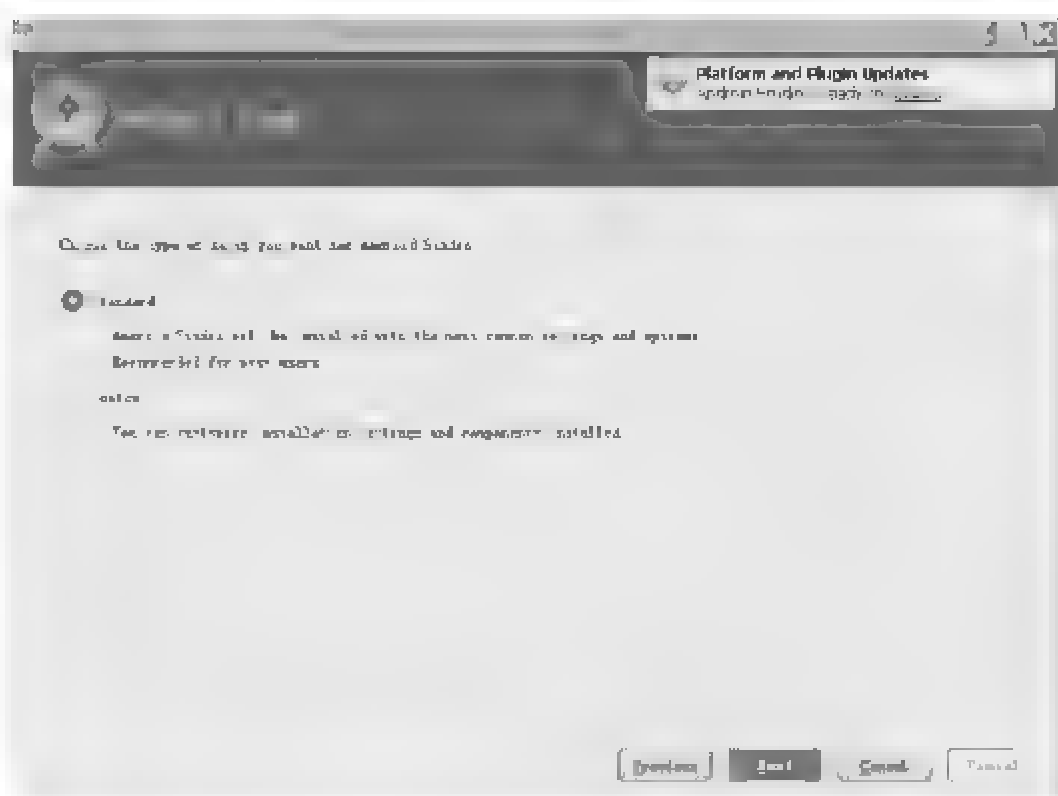


图 1-6　安装向导二

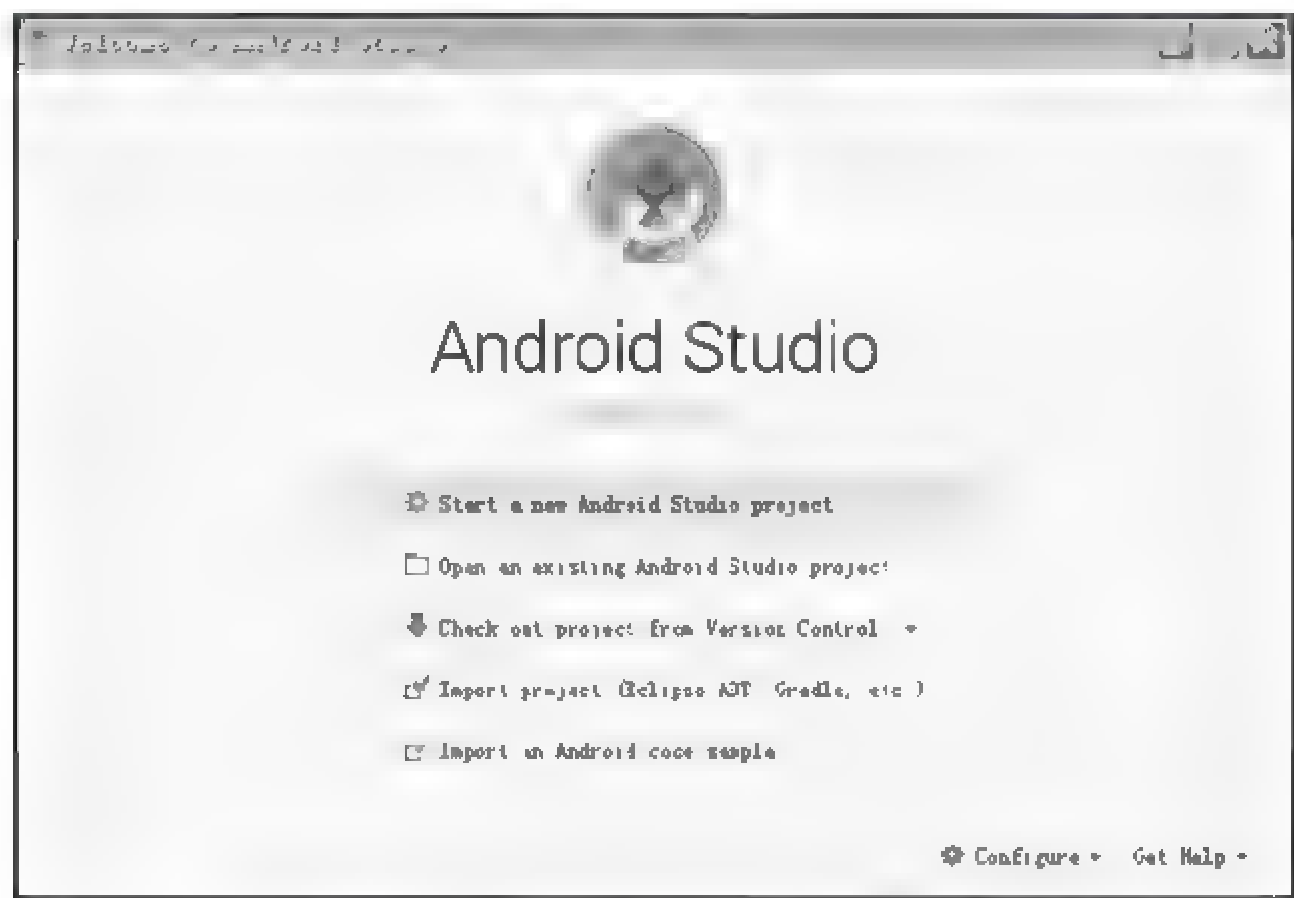


图 1-7　Android Studio 的欢迎界面

注意，配置过程可能发生如下错误提示：

（1）配置过程中提示 Your Android SDK is missing...，请检查 SDK 的安装路径是否正确配置，同时检查环境变量中系统变量的 ANDROID_HOME 是否正确设置。

（2）第一次打开 Android Studio 可能会报错 Unable to access Android SDK add-on list，这个界面不用理会，单击 Cancel 按钮即可。进入 Android Studio 主界面后，依次选择菜单 File→Project Structure→SDK Location，在弹出的窗口中分别设置 JDK、SDK、NDK 的路径。设置完毕后再打开 Android Studio 就不会报错了。

（3）已经按照安装步骤正确安装，运行 Android Studio 却总是打不开。请检查电脑上是否开启了防火墙，建议关闭系统防火墙及所有杀毒软件的防火墙。关了防火墙后再重新打开 Android Studio 试试。

Android Studio 2.2.3 完整版安装的 SDK 只有 Android7.1.1(API 25)和 25.0.1 版本的编译工具集合。虽然有一个版本的 SDK 就足够应付开发和编译，但是企业开发时需要同时运用多种版本，以便测试 App 在不同机型、不同版本上的兼容性，因此建议同时安装几个常用的 SDK 版本。



运行 SDK 安装目录下的 SDK Manager.exe，弹出的窗口显示除了 API 25 版本外，其他版本的 SDK 与编译工具均未安装，如图 1-8 所示。

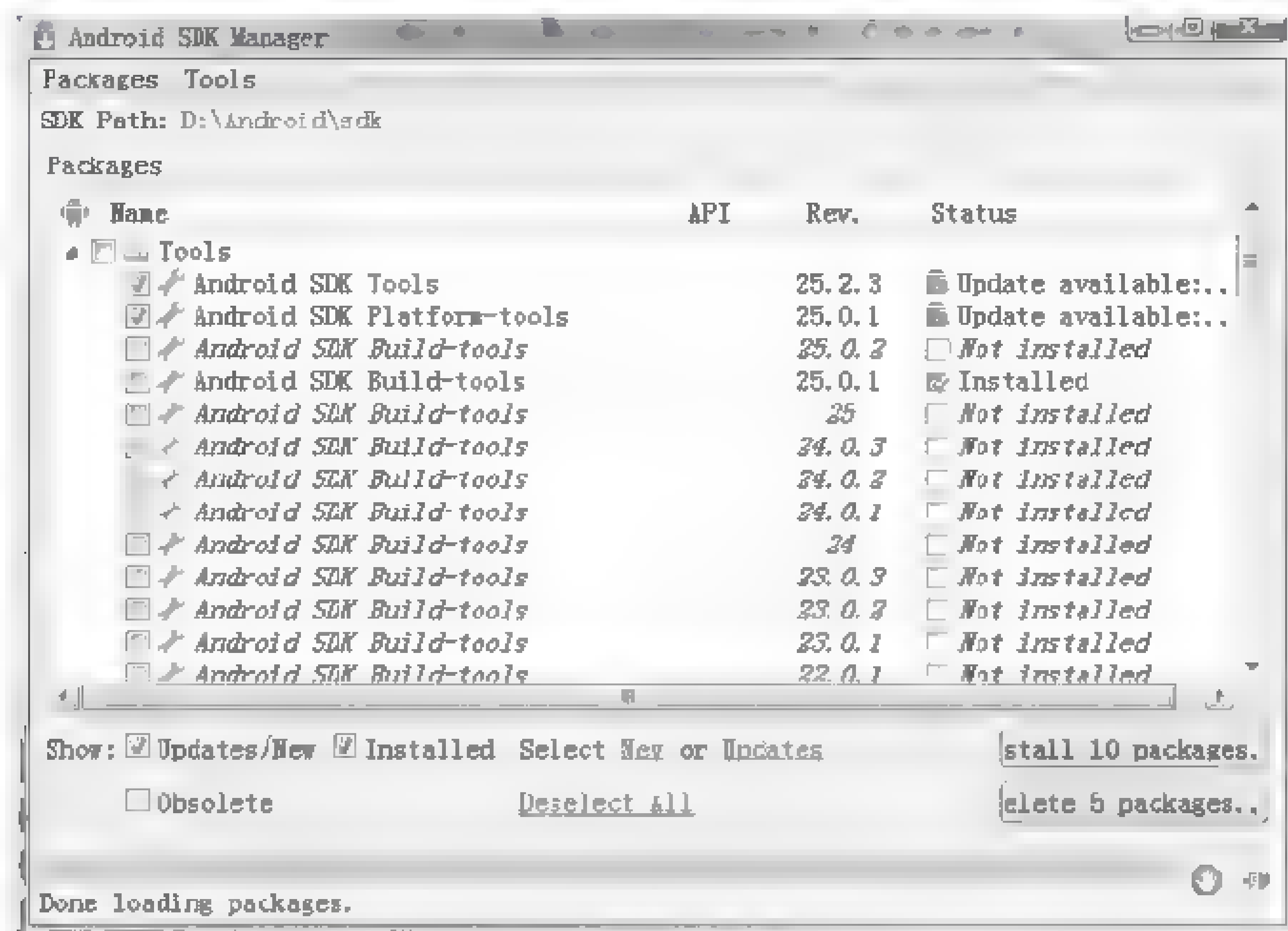


图 1-8 默认安装的 SDK Manager 初始界面

这里要勾选 Tools 复选框，拉下来勾选 Android7.0(API 24)的 SDK Platform、Android6.0(API 23)的 SDK Platform、Android5.1.1(API 22)的 SDK Platform、Android5.0.1(API 21)的 SDK Platform、Android4.4.2(API 19)的 SDK Platform，然后单击右下角的"Install ** packages..."按钮，耐心等待安装更新。

1.3 运行小应用 Hello World

成功安装 Android Studio 后，打开其界面会发现有一堆菜单和图标，对于这个陌生的开发环境，读者可能会有不知所措的感觉。现在不逐一讲解每个菜单和图标的作用，直接开始第一个 App——Hello World，让我们在实践中边学边用，更好地理解和吸收。

1.3.1 创建新项目

打开 Android Studio，依次选择菜单 File→New→New Project，弹出 Create New Project 窗口，如图 1-9 所示。在 Application name 栏输入应用名称，在 Company Domain 栏输入公司域名，下面会自动合成工程的包名，选择好项目工程的保存目录，单击 Next 按钮。

下一个界面是目标设备界面，如图 1-10 所示。该界面可选择 App 期望运行在什么设备上，以及运行 App 所需的 SDK 最低版本号，Minimun SDK 右下方的文字提示当前版本号支持的设备市场份额。这里不做变动，按照默认勾选的 Phone and Tablet 即可，最低版本号也是默认的 API 15（支持设备的市场份额为 97.3%，能够满足绝大部分机型）。然后单击 Next 按钮，进入下一个界面，如图 1-11 所示。该界面提示我们选择初始界面风格，这里还是保持默认的选项 Empty Activity，单击 Next 按钮。

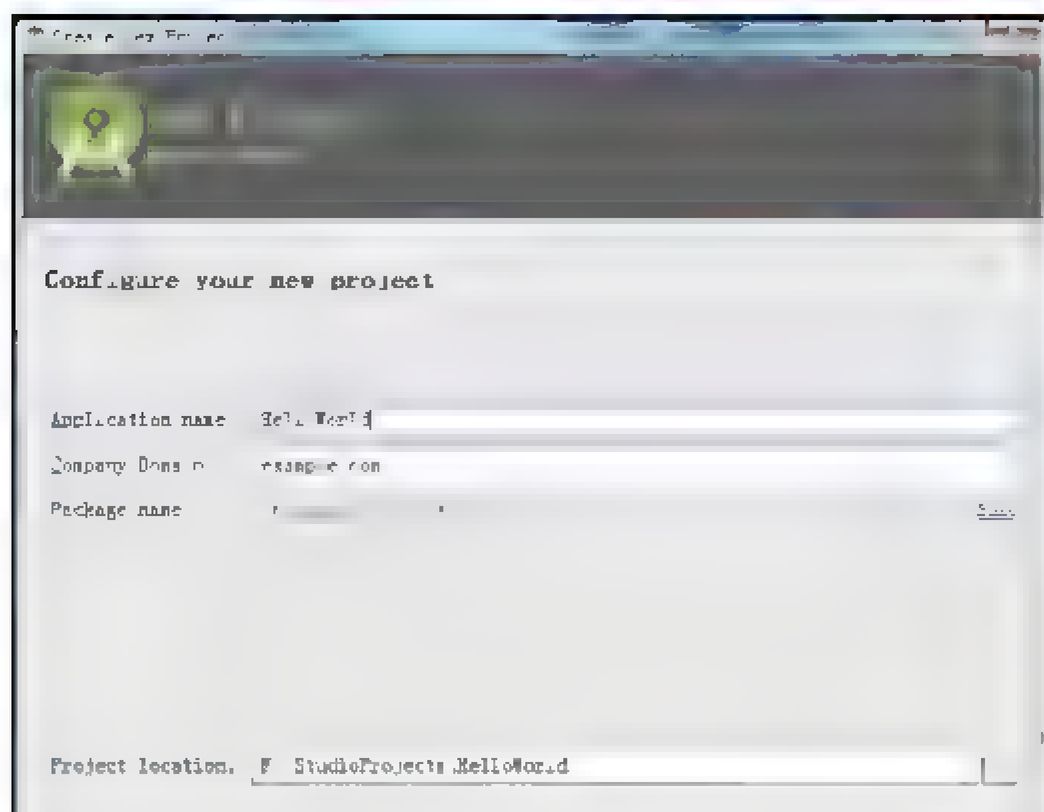


图 1-9　创建新项目

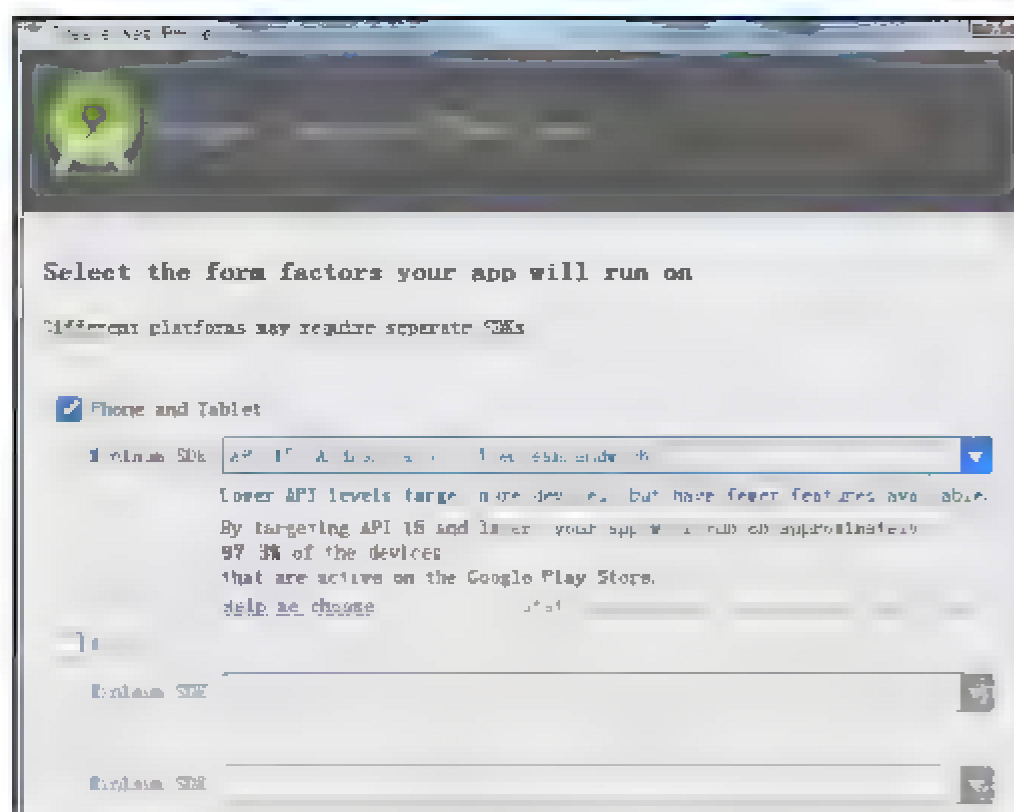


图 1-10　指定目标设备

下一个界面是入口设置界面，如图 1-12 所示。该界面可输入活动名称（Activity Name）与布局名称（Layout Name），正常情况使用默认名称即可，单击 OK 按钮，等待工程创建。

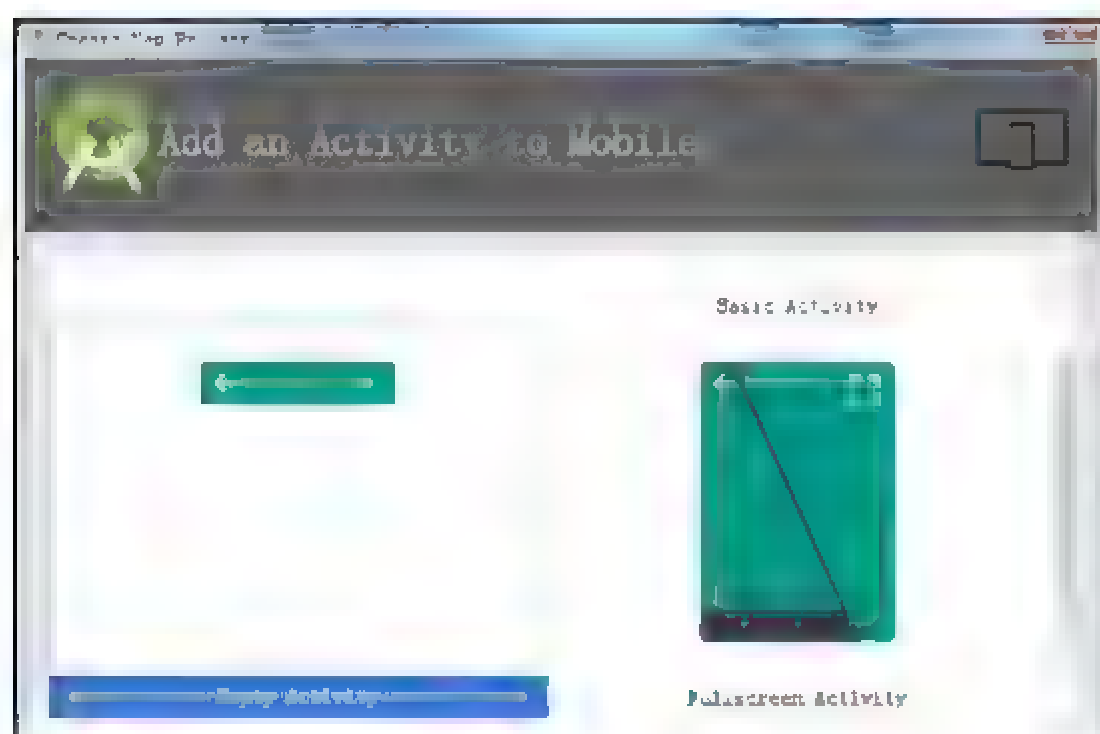


图 1-11　指定 Activity 界面的风格



图 1-12　设置入口界面的名称

工程创建完毕后，Android Studio 自动打开 activity_main.xml 与 MainActivity.java，并默认展示 MainActivity.java 的源码，如图 1-13 所示。

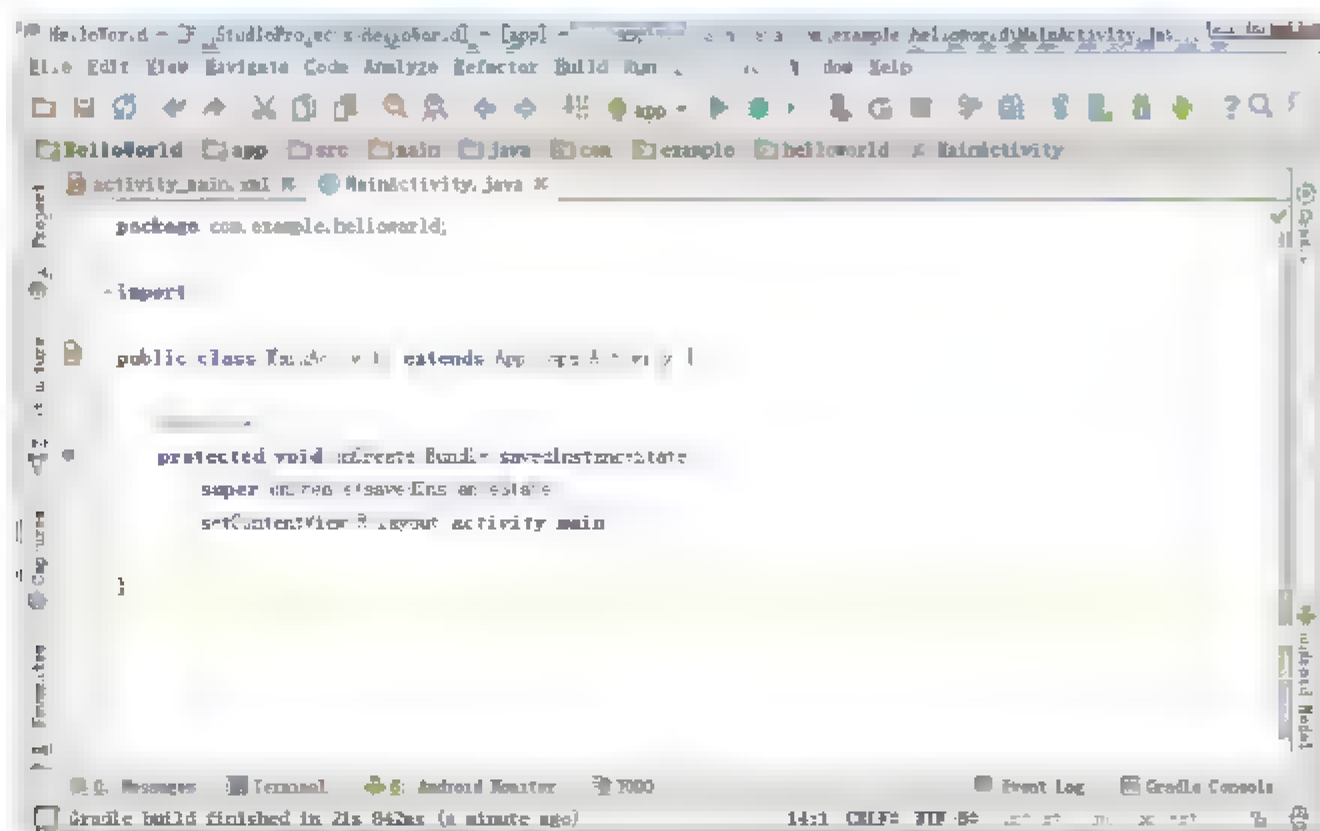


图 1-13　默认创建的 MainActivity

MainActivity.java 上方的标签表示该文件的路径结构，注意源码左侧有一列标签，从上到下依次是 Project、Structure、Captures、Favorites。单击 Project 标签，左侧会展开小窗口表示该项目工程的目录结构，如图 1-14 所示。单击 Structure 标签，左侧会展开小窗口表示该代码的内部方法结构，如图 1-15 所示。

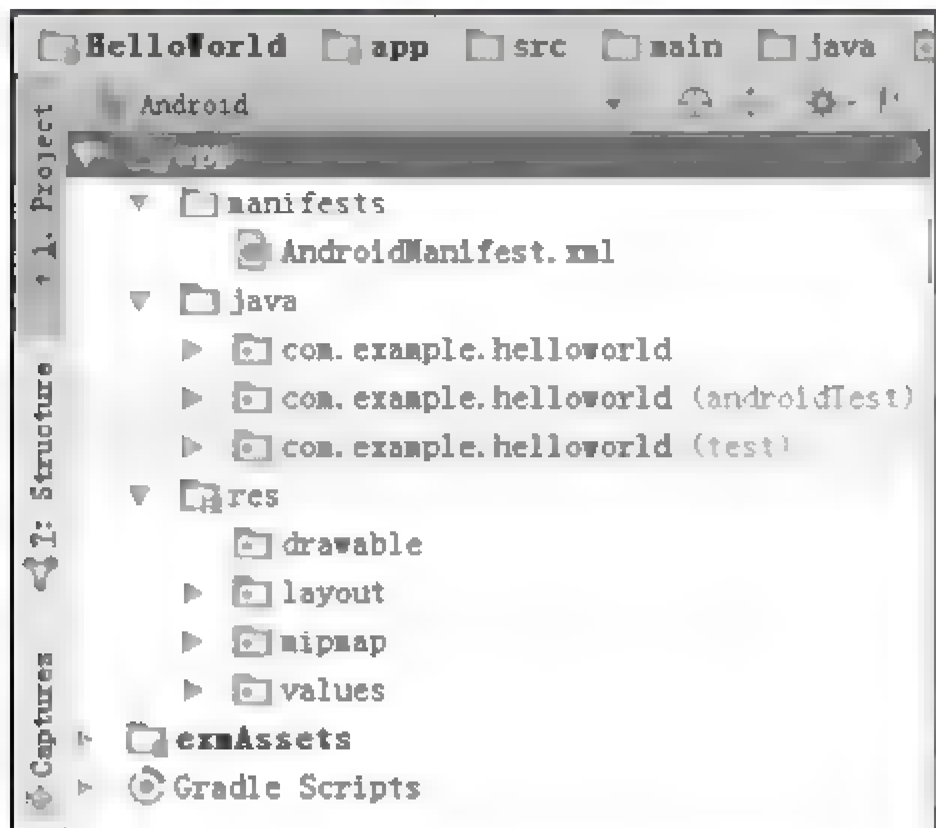


图 1-14　HelloWorld 的工程结构



图 1-15　MainActivity 的方法结构

看完代码文件再来看布局文件，单击 activity_main.xml 标签，切换到布局文件设计展示界面，如图 1-16 所示。可以看到左侧多了一列 Palette 窗口，内部是各种布局与控件列表。在 Palette 窗口下方有两个标签，分别是 Design（默认选中，表示设计图）和 Text（表示源代码）。单击 Text 标签，切换到布局文件的源码界面，如图 1-17 所示。这个布局文件是标准的 XML 格式，内部定义了 App 页面上包含的各种控件元素及其排列组合方式。

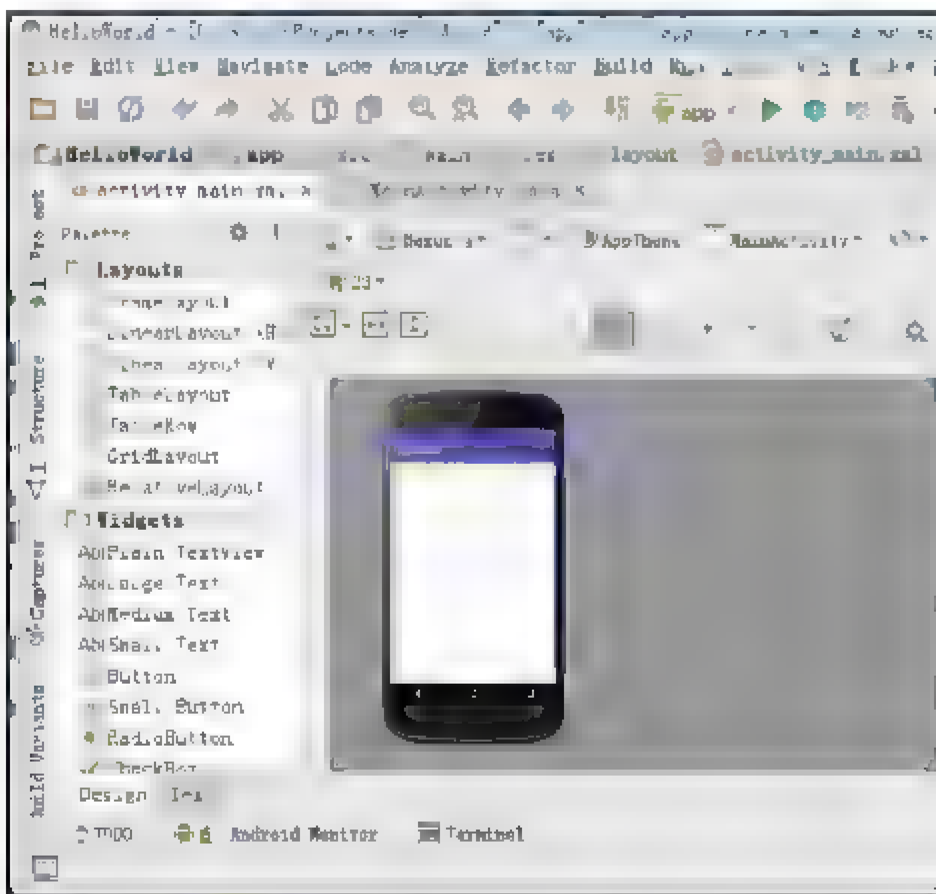


图 1-16　activity.xml 的设计图

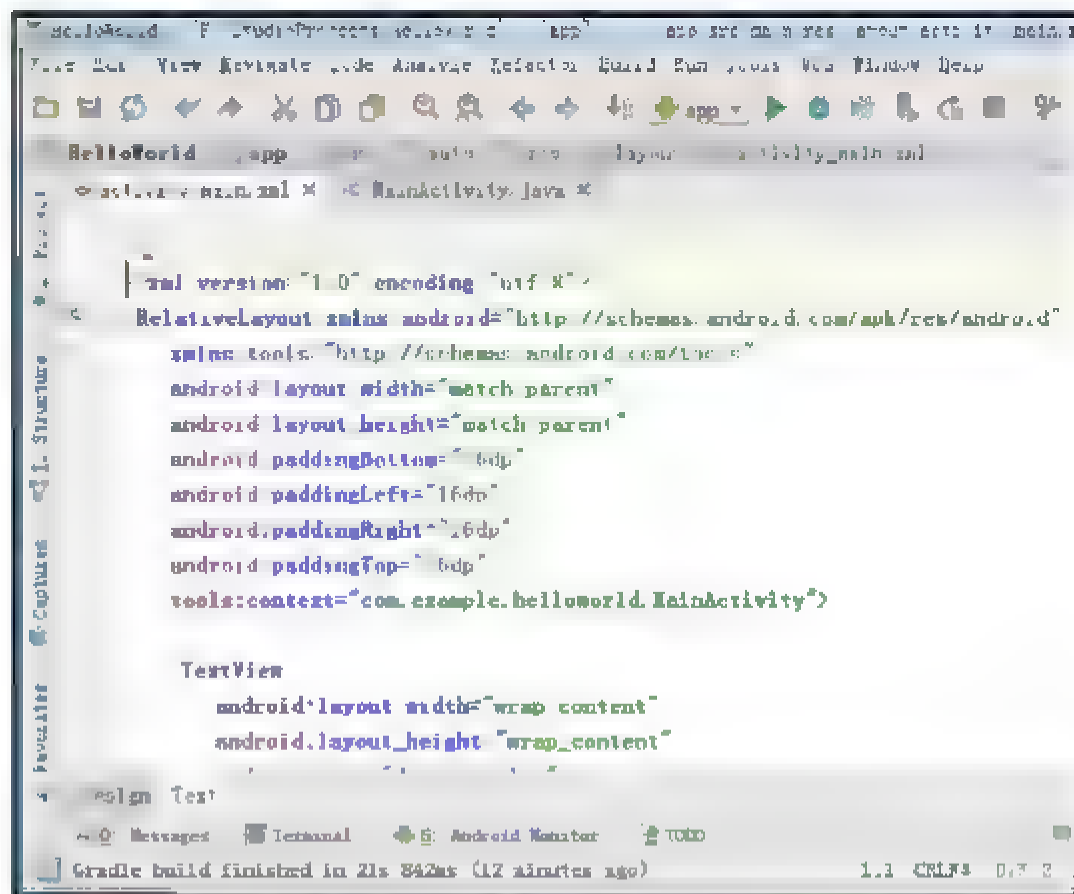


图 1-17　activity.xml 的源代码

在查看 activity_main.xml 的设计图时，可能会遇到以下问题：

（1）报错 Android N requires the IDE to be running with Java 1.8 or later。

原因是当前机器的 jdk 版本低于 1.8（如 jdk1.6 或 jdk1.7）。

解决办法：安装最新的 jdk1.8，并正确配置 JDK 的安装路径。

（2）报错 Rendering Problems Exception raised during rendering：com/android/util/PropertiesMap (Details) 或 Rendering Problems Exception raised during rendering：com.android.ide.common.rendering.api.LayoutlibCallback。

原因不明，可能是 Android Studio 的一个 bug。

解决办法：把布局设计图右上方的编译器改为 API 23，如图 1-18 所示。

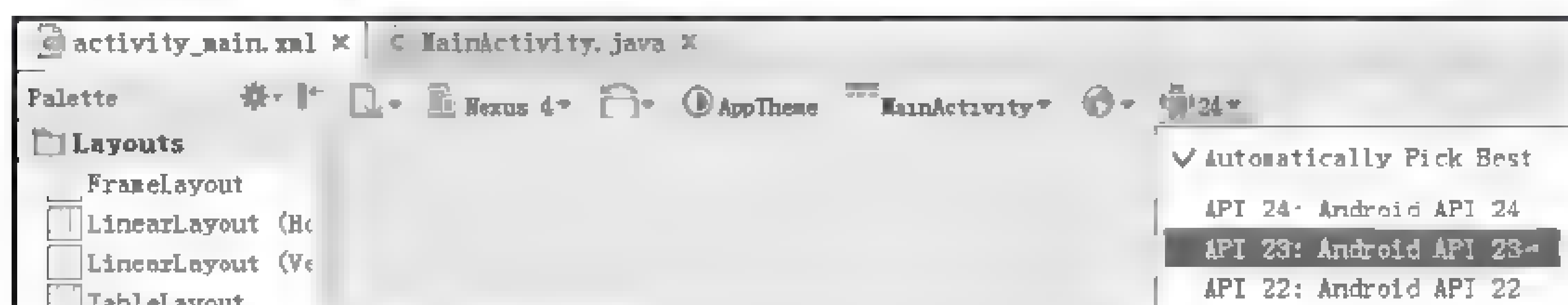


图 1-18　更改布局设计图的编译器版本

1.3.2　编译项目/模块

Android Studio 与 Eclipse 一样，如果代码没有报错，Android Studio 就会自动编译，我们只需直接运行项目即可。当然有时候我们想手动重新编译，有以下 3 种编译方式：

（1）选择菜单 Build→Make Project，编译整个项目下的所有模块。

（2）选择菜单 Build→Make Module ***，编译指定名称的模块。

（3）选择菜单 Build→Clean Project，然后选择菜单 Build→Rebuild Project，先清理项目，再对整个项目重新编译。

下面先认识一下任务栏上的几个常用图标，后面会经常用到它们。

在图 1-19 中，第 4 个竖屏图标是 AVD Manager 按钮，单击该按钮会弹出模拟器的管理窗口；倒数第二个向下箭头图标是 SDK Manager，单击该按钮会弹出 SDK 版本的管理窗口。

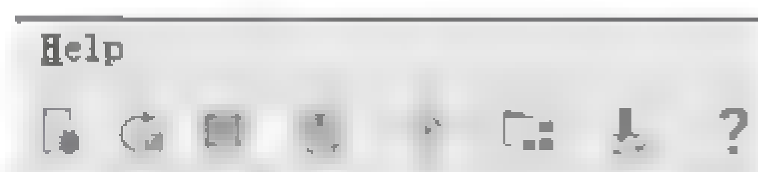


图 1-19　任务栏上的常用图标

1.3.3　创建模拟器

所谓模拟器，是指在电脑上构造一个演示窗口，模拟手机屏幕上的 App 运行效果。App 通过编译后，要选择一个接入设备来运行，依次选择菜单 Run→Run 'app'（也可按快捷键 Shift+F10），Android Studio 会弹出新窗口 Select Deployment Target，如图 1-20 所示。

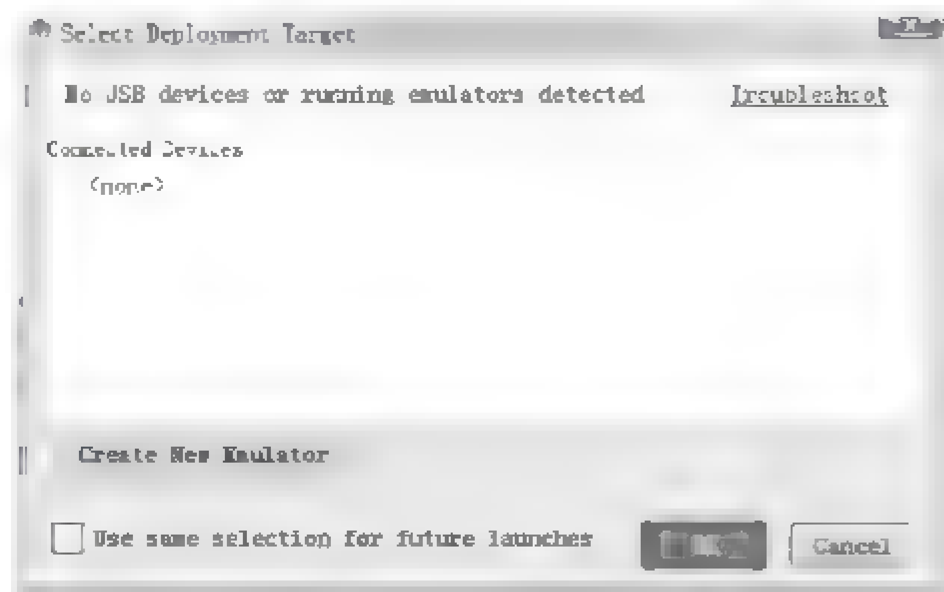


图 1-20　运行 App 选择接入设备

对初学者来说，一开始没有可用的模拟器，得创建新模拟器，单击 Create New Emulator 按钮，弹出模拟器的配置界面，如图 1-21 所示。按照默认配置即可，单击 Next 按钮。

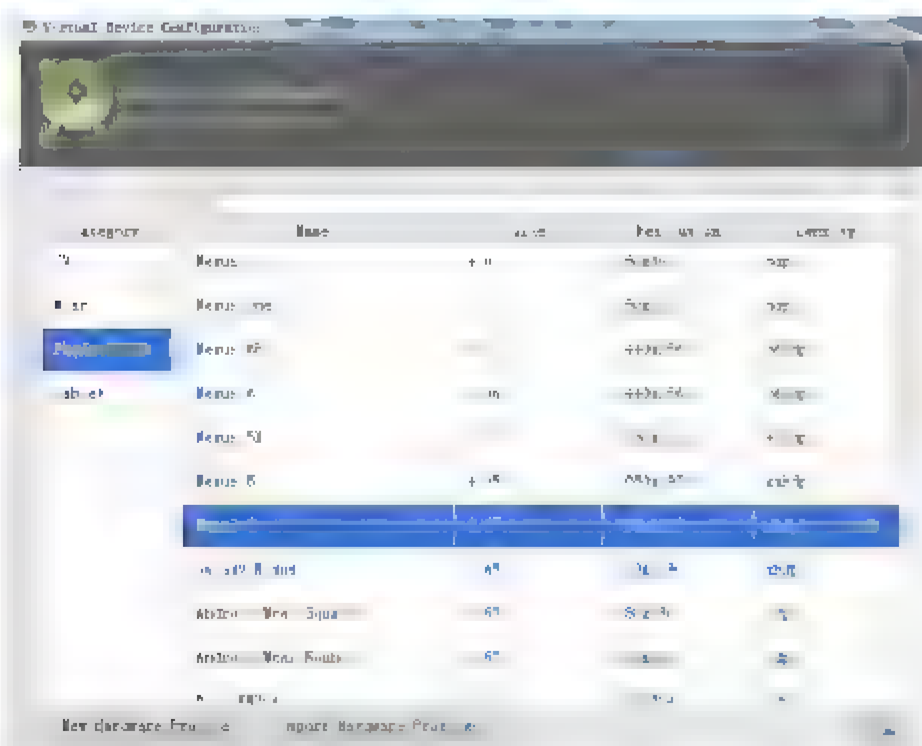


图 1-21 选择模拟器的分辨率

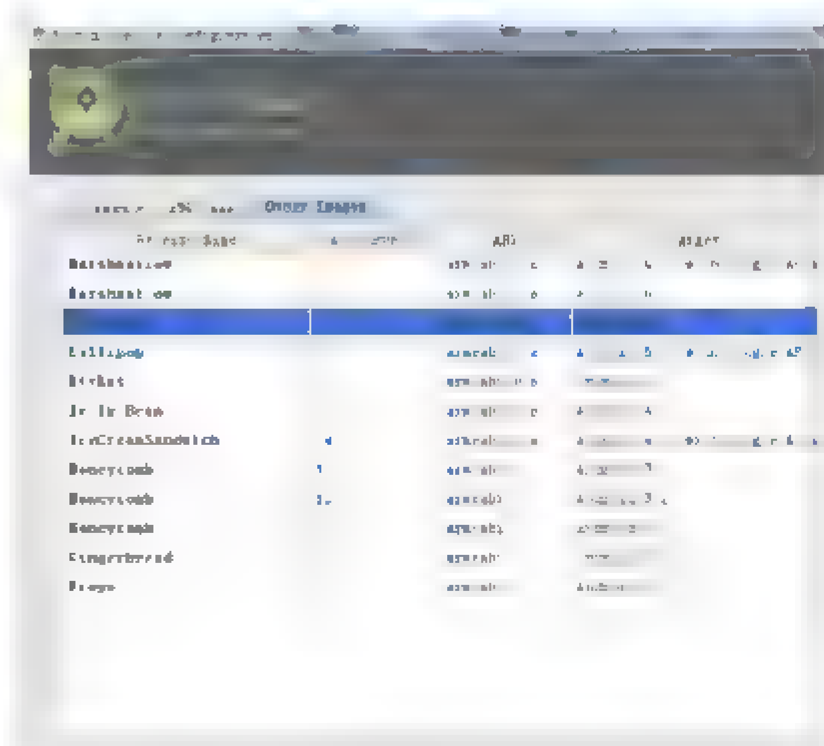


图 1-22 选择模拟器的 SDK 版本

下一个界面是 SDK 版本的选择界面，如图 1-22 所示。单击第 3 个标签 Other Images，在列表中选择第一个 Lollipop（即 Android 5.1），表示接下来创建的模拟器是基于 Android 5.1 系统的。如果读者的开发机配置不是很优越，那么建议模拟器的大小不超过 5 寸，同时 SDK 版本不高于 API 19，因为分辨率越大、版本越高，消耗的系统资源也越大。

单击 Next 按钮，进入最后的确认界面，在确认界面右下角单击 Finish 按钮，等待模拟器的创建。

1.3.4 在模拟器上运行 App

模拟器创建完成后，重新依次选择菜单 Run→Run 'app'，这时弹出的窗口中会出现刚才创建的模拟器，名称为 Nexus 4 API 22，如图 1-23 所示。

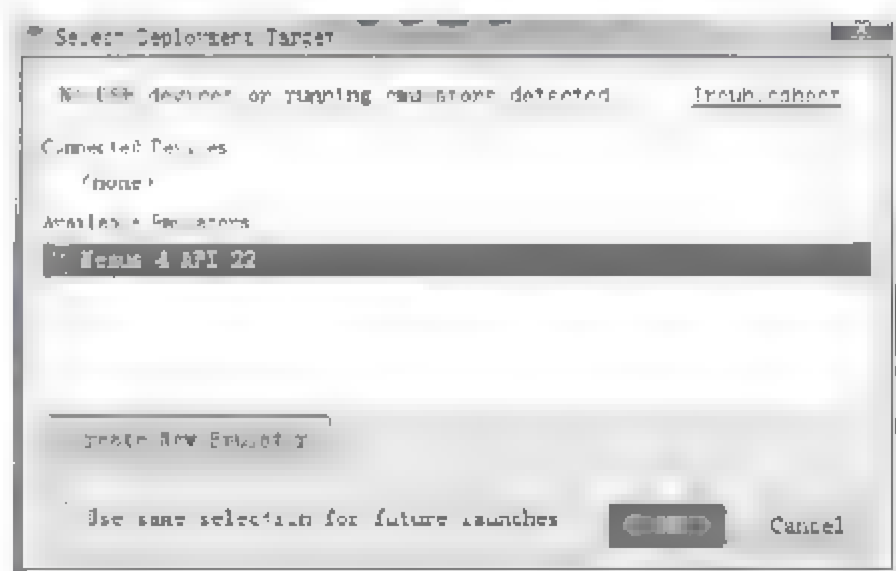


图 1-23 接入设备界面出现新创建的模拟器

选中该模拟器，单击 OK 按钮，等待 Android Studio 启动模拟器。关于模拟器的启动结果，可以查看主界面下方的提示窗口，如图 1-24 所示。提示窗口有左右两个小窗口，左侧窗口的左上角有一个 logcat 标签，用于展示 App 的运行日志；右侧窗口的右下角有一个 Gradle Console 标签，用于展示 App 工程的编译与启动情况。

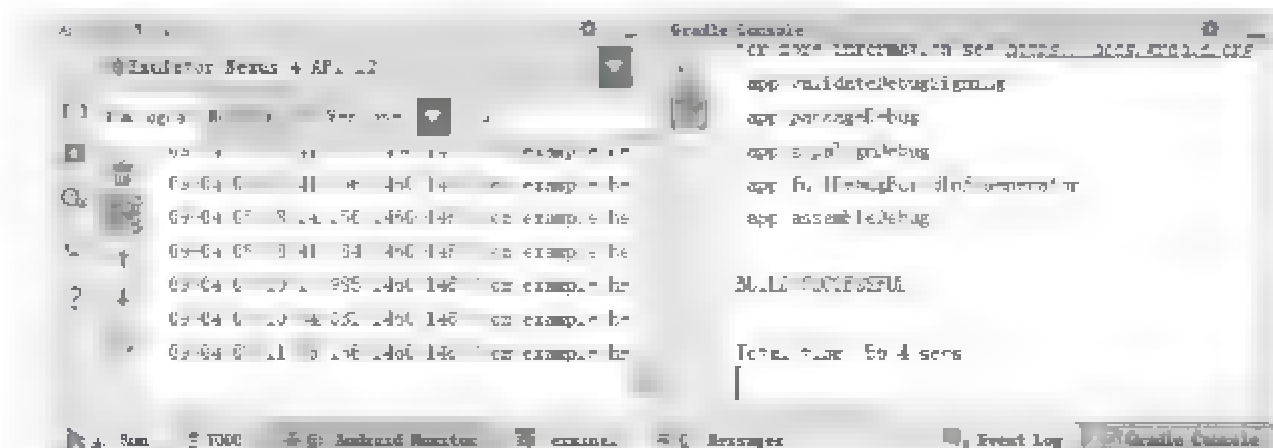


图 1-24 App 运行结果跟踪窗口

如果在 Gradle Console 窗口提示编译或启动失败，就按照提示信息进行处理。如果 Gradle Console 窗口提示成功，等待模拟器启动完成后，就会出现类似手机的模拟器界面，如图 1-25 所示。把模拟器屏幕下方中间的解锁图像向上拖动，使得屏幕解锁成功，这时进入 App 的启动界面 Hello World，如图 1-26 所示。



图 1-25　模拟器启动完成屏幕



图 1-26　HelloWorld 的启动界面

如果 App 启动界面正常展示，那么恭喜你，第一个 Hello World App 就这样成功了。都说万事开头难，前面我们经过各种困难，终于搭建好 Android Studio 的开发环境，并且成功运行了第一个 App——Hello World，不过这只是万里长征的第一步，接下来还有更奇妙的 Android 世界等着我们去探索。

1.4　App 的工程结构

上一节我们在模拟器上成功地运行了第一个 App（Hello Wolrd），接下来好好研究一下它的工程结构。每个 App 的工程结构都差不多，只要掌握了基本结构，后面开发起来就会得心应手。

1.4.1　工程目录说明

Android Studio 的工程创建分两个层级：第一个层级通过菜单 File→New→New Project 创建，这里的新项目是指新的工作空间，对应 Eclipse 的 workspace；第二个层级通过菜单 File→New→New Module 创建，这里的新模块是指一个单独的 App 工程，对应 Eclipse 的 project。第一次运行 Android Studio 都是选择 New Project，表示先创建一个工作空间；后面还想创建新的 App 工程时，只需选择 New Module，表示在当前工作空间下新建一个 App 工程。

例如，图 1-27 是之前 HelloWorld 工程的目录结构图。



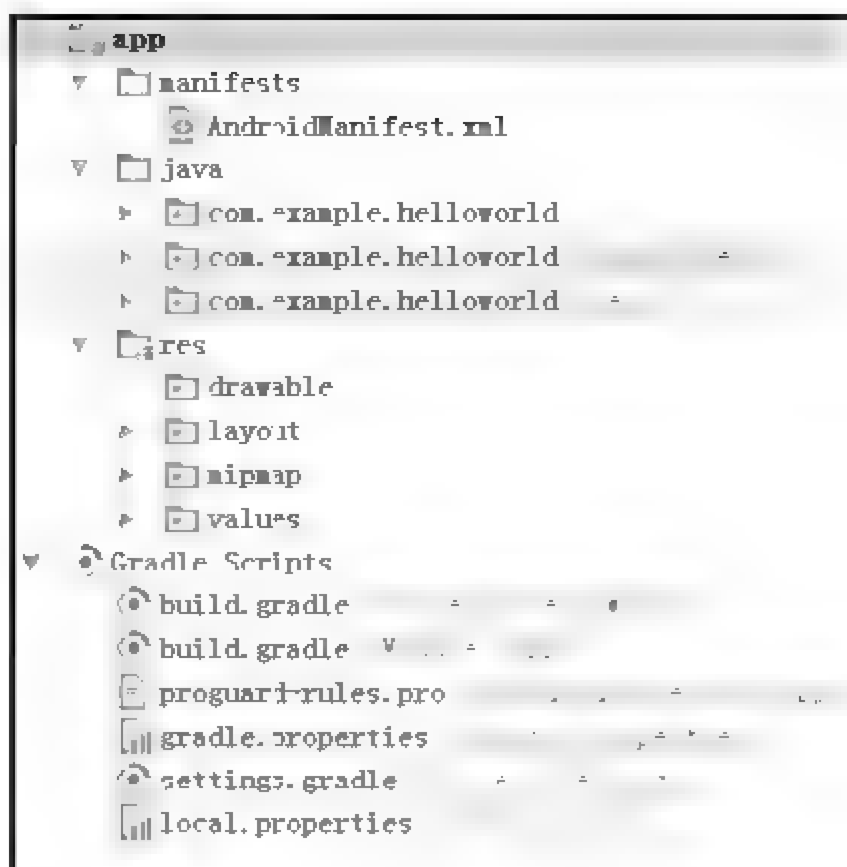


图 1-27　HelloWorld 工程的目录结构图

从结构图中可以看到，该工程下面有两个目录：一个是 app，另一个是 Gradle Scripts。其中，app 下面又有 3 个子目录，功能说明如下：

（1）manifests 子目录，下面只有一个 xml 文件，即 AndroidManifest.xml，是 App 的运行配置文件。

（2）java 子目录，下面有 3 个 com.example.hellorworld 包，其中第一个包存放的是 App 工程的 java 源代码，后面两个包存放的是测试用的 java 代码。

（3）res 子目录，存放的是 App 工程的资源文件。res 子目录下又有 4 个子目录：

- drawable 目录存放的是图形描述文件与用户图片。
- layout 目录存放的是 App 页面的布局文件。
- mipmap 目录存放的是启动图标。
- values 目录存放的是一些常量定义文件，比如字符串常量 strings.xml、像素常量 dimens.xml、颜色常量 colors.xml、样式风格定义 styles.xml 等。

Gradle Scripts 下面主要是工程的编译配置文件，主要有：

（1）build.gradle，该文件分为项目级与模块级两种，用于描述 App 工程的编译规则。

（2）proguard-rules.pro，该文件用于描述 java 文件的代码混淆规则。

（3）gradle.properties，该文件用于配置编译工程的命令行参数，一般无须改动。

（4）settings.gradle，配置哪些模块在一起编译。初始内容为 include ':app'，表示只编译 App 模块。

（5）local.properties，项目的本地配置，一般无须改动。该文件是在工程编译时自动生成的，用于描述开发者本机的环境配置，比如 SDK 的本地路径、NDK 的本地路径等。

1.4.2　编译配置文件 build.gradle

项目级别的 build.gradle 一般无须改动，我们只需关注模块级别的 build.gradle。下面在初始的 build.gradle 文件中补充文字注释，方便读者更好地理解每个参数的用途。

```
apply plugin: 'com.android.application'

android {
    // 指定编译用的 SDK 版本号，如 25 表示使用 Android 7.1 编译
    compileSdkVersion 25
    // 指定编译工具的版本号。这里的头两位数字必须与 compileSdkVersion 保持一致，具体的版本号可在 sdk 安装目录的 sdk\build-tools 下找到
    buildToolsVersion "25.0.2"

    defaultConfig {
        // 指定该模块的应用编号，即 App 的包名。该参数为自动生成，无须修改
        applicationId "com.example.helloworld"
        // 指定 App 适合运行的最小 SDK 版本号，如 15 表示至少要在 Android 4.0.3 上运行
        minSdkVersion 15
        // 指定目标设备的 SDK 版本号，即该 App 最希望在哪个版本的 Android 上运行
        targetSdkVersion 25
        // 指定 App 的应用版本号
        versionCode 1
        // 指定 App 的应用版本名称
        versionName "1.0"
    }
    buildTypes {
        release {
            // 指定是否开启代码混淆功能。true 表示开启混淆，false 表示无须混淆。
            minifyEnabled false
            // 指定代码混淆规则文件的文件名
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

// 指定 App 编译的依赖信息
dependencies {
    // 指定引用 jar 包的路径
    compile fileTree(dir: 'libs', include: ['*.jar'])
    // 指定单元测试编译用的 junit 版本号
    testCompile 'junit:junit:4.12'
    // 指定编译 Android 的高版本支持库，如 AppCompatActivity 必须指定编译 appcompat-v7 库
    compile 'com.android.support:appcompat-v7:25.1.0'
}
```

1.4.3 App 运行配置 AndroidManifest.xml

AndroidManifest.xml 用于指定 App 内部的运行配置，是一个 XML 描述文件，根节点为 manifest，根节点的 package 指定了该 App 的包名。manifest 下面又有若干子节点，分别说明如下：

（1）uses-sdk，该节点有两个属性：android:minSdkVersion 和 android:targetSdkVersion。这两个属性是早期 Eclipse 开发 App 时使用的，现在这两个字段改成放到 build.gradle 文件中，故而 Android Studio 不配置 uses-sdk 也没有关系。

（2）uses-permission，该节点用于声明 App 运行过程中需要的权限名称。例如，访问网络需要上网权限，拍照需要摄像头权限，定位需要定位权限等。

（3）application，该节点用于指定 App 的自身属性，默认的属性说明如下：

- android:allowBackup，用于指定是否允许备份，开发阶段设置为 true，上线时设置为 false。
- android:icon，用于指定该 App 在手机屏幕上显示的图标。
- android:label，用于指定该 App 在手机屏幕上显示的名称。
- android:supportsRtl，设置为 true 表示支持阿拉伯语/波斯语这种从右往左的文字排列顺序。
- android:theme，用于指定该 App 的显示风格。

application 节点下还有几个子节点，比如活动 activity、服务 service、广播接收器 receiver、内容提供器 provider 等，这些子节点的详细属性会在后续章节详细说明。

1.4.4 在代码中操纵控件

在一开始创建 Hello World 工程时，Android Studio 默认打开了两个文件，分别是布局文件 activity.xml 和代码文件 MainActivity.java。下面先看布局文件 activity.xml 的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.helloworld.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
</RelativeLayout>
```

这里可以看到 xml 文件中只有两个节点，分别是 RelativeLayout 和 TextView。再仔细看看，有没有发现我们熟悉的 Hello World？没错，模拟器 App 界面显示的 Hello World 就来自于这里，也就是 TextView 控件的 android:text 属性值。可以把这里的 Hello World 改为其他文字，比如“你好、世界”或 I Love Android，改完保存文件后再依次选择菜单 Run→Run 'app'，看看 App 界面上的文字是不是变成新的了？

当然，我们的目标并不仅限于在布局文件中修改文字，还要能够在代码中修改文字的内容。再次打开代码文件 MainActivity.java，看看里面有什么内容。该 java 文件中 MainActivity 类的内容如下：

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

这里可以看出，MainActivity.java 的代码内容很简单，只有一个 MainActivity 类，该类下面只有一个函数 onCreate。注意 onCreate 内部的 setContentView 方法直接引用了布局文件的名字 activity_main，该方法的意思是往 App 界面填充 activity.xml 的布局内容。现在我们要在这里改动改动，加点“绿叶红花”让它好看一些。首先打开 activity.xml，在 TextView 节点下方补充一行 android:id="@+id/tv_hello"；然后回到 MainActivity.java，在 setContentView 方法下面补充如下几行代码：

```
        //获取名字为 tv_hello 的 TextView 控件
        TextView tv_hello = (TextView) findViewById(R.id.tv_hello);
        //给 TextView 控件设置文字内容
        tv_hello.setText("今天天气真热啊，火辣辣的");
        //给 TextView 控件设置文字颜色
        tv_hello.setTextColor(Color.RED);
        //给 TextView 控件设置文字大小
        tv_hello.setTextSize(30);
```

保存文件后依次选择菜单 Run→Run 'app'，模拟器上的 App 界面就变成了如图 1-28 所示的样子。

现在不但文字内容改变了，文字颜色和字体大小也发生了变化。怎么样，是不是很有成就感呢？好的开始是成功的一半，现在我们初步学会了在代码中操作控件，下一章进一步学习在 App 界面上人机交互。





图 1-28 修改文字后的 HelloWorld 界面

1.5 准备开始

俗话说得好，磨刀不误砍柴工。尽管前面我们已经初步学会了通过代码操作控件，不过为了后面介绍 Android 更顺利些，建议读者先了解本节的准备工作。如果读者已经迫不及待要进入 Android 的开发世界，也可以暂时跳过本节直接翻到第 2 章，符合个人习惯就好。

1.5.1 使用快捷键

就像在 Eclipse 上进行 java 开发一样，善用快捷键会让开发者提高工作效率，Android Studio 也是一样，下面是使用 Android Studio 开发 App 常用的快捷键。

- Ctrl+S：保存文件。
- Ctrl+Z：撤销上次的编辑。
- Ctrl+Shift+Z：重做上次的编辑，建议改为 Ctrl+Y，与 Eclipse、UEStudio 等工具保持一致。Android Studio 默认 Ctrl+Y 为删除当前行，这点不太好，当你习惯按 Ctrl+Y 重做上次编辑时，系统却删除了当前行，非常不便。
- Ctrl+C：复制。
- Ctrl+X：剪切。
- Ctrl+V：粘贴。
- Ctrl+A：全选。
- Delete：删除。
- Ctrl+F：查询。
- Ctrl+R：替换。

- Ctrl+/：注释选中代码（在每行代码前面加双斜杆）。
- Ctrl+Shift+/：注释选中的代码段（在选中的代码段前面加“/*”，后面加“*/”）。
- Ctrl+Alt+L：格式化选中的代码段。注意该快捷键与 QQ 默认的热键（锁定 QQ）冲突，建议更换快捷键，或者删除 QQ 的同名热键。
- Shift+F6：重命名。建议改为 F2，与 Wnidows 和 Eclipse 的使用习惯保持一致。
- Alt+Enter：给光标所在位置的类导入相应的包。
- Shift+F10：运行当前模块。
- Ctrl+F5：清理并重新运行当前模块。

当然，每个人习惯的快捷键不尽相同，对于 Android Studio 来说也不例外，为了更好地使用快捷键，最好手工修改快捷键。手工修改快捷键的方法：依次选择菜单 File→Settings，在弹出的设置窗口中选择 Keymap，窗口右侧出现如图 1-29 所示的快捷键列表。

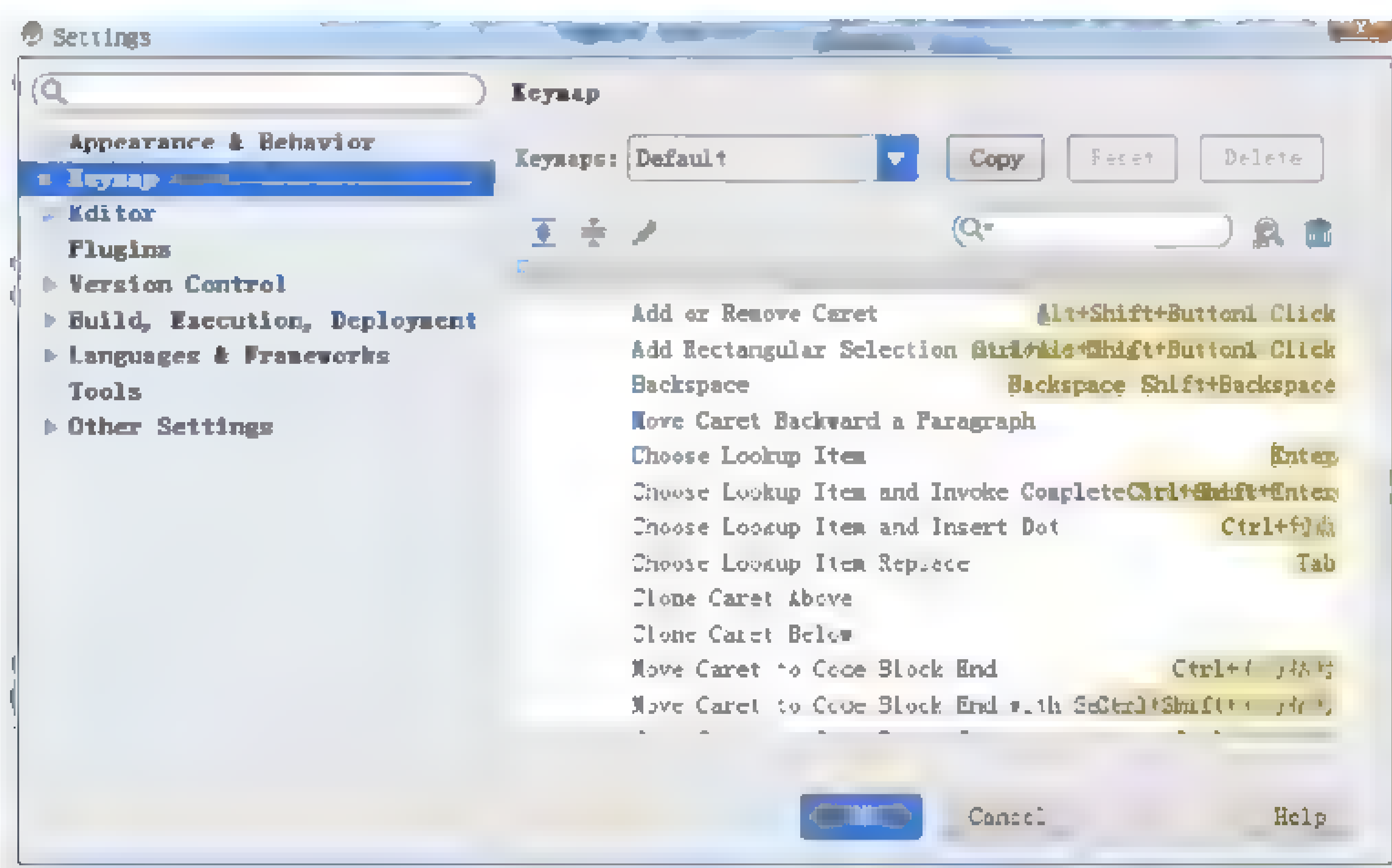


图 1-29　快捷键设置界面

在设置界面选中某条快捷键，右击或单击上方的铅笔按钮，在弹出的菜单中选择 Add Keyboard Shortcut，然后在键盘上按你要设置的快捷键组合，单击 OK 按钮，即可完成对应的快捷键设置。

1.5.2　安装 SVN 工具

在企业里面开发 App 都是团队合作，需要对代码进行统一管理，而且 App 每隔一两周便发布一个新版本，这也要求做好工程代码的版本控制。因此，企业开发 App 都会运用版本控制工具管理工程源码，最常见的版本控制工具是 SVN。

Android Studio 自带了 SVN 插件（Subversion），但是还需要开发者进行相关配置才能正常使用 SVN 功能。具体配置步骤如下：

步骤 01 在本机上安装 TortoiseSVN。

首先下载 TortoiseSVN 安装包，然后在安装时选择 command line client tools，这样安装后在 bin 目录下才能找到命令行工具 svn.exe。



步骤02 在 Android Studio 中配置 TortoiseSVN 的命令行工具。

打开 Android Studio，依次选择菜单 File→Settings→Version Control→Subversion→user command line client，单击右侧的浏览按钮，选择本地安装的 svn.exe 的完整路径。

步骤03 在 Android Studio 中使用 SVN 检出项目。

打开 Android Studio，依次选择菜单 VCS→Checkout from Version Control→Subversion，单击 Repositories 右方的加号按钮，在弹出的小窗口中输入 SVN 仓库地址，单击 OK 按钮，回到原窗口单击 Checkout 按钮，把项目检出到本地目录。

项目检出完毕后，在开发过程中要及时把改好的代码提交到 SVN，同时要及时从 SVN 更新别人改过的代码到本地。下面是 SVN 更新/提交的方法：

（1）把代码提交给 SVN 服务器：选中并右击工程目录，依次选择菜单 Subversion→Commit File...，表示向 SVN 服务器提交本地改过的文件。

（2）从 SVN 服务器更新代码：选中并右击工程目录，依次选择菜单 Subversion→Update File...，表示从 SVN 服务器更新文件到本地目录。

1.5.3 安装常用插件

在 Android Studio 中安装插件的步骤与 Eclipse 类似，具体步骤为：依次选择菜单 File→Settings→Plugins→下方按钮 Browser repositories...，弹出当前可用插件列表窗口，如图 1-30 所示。

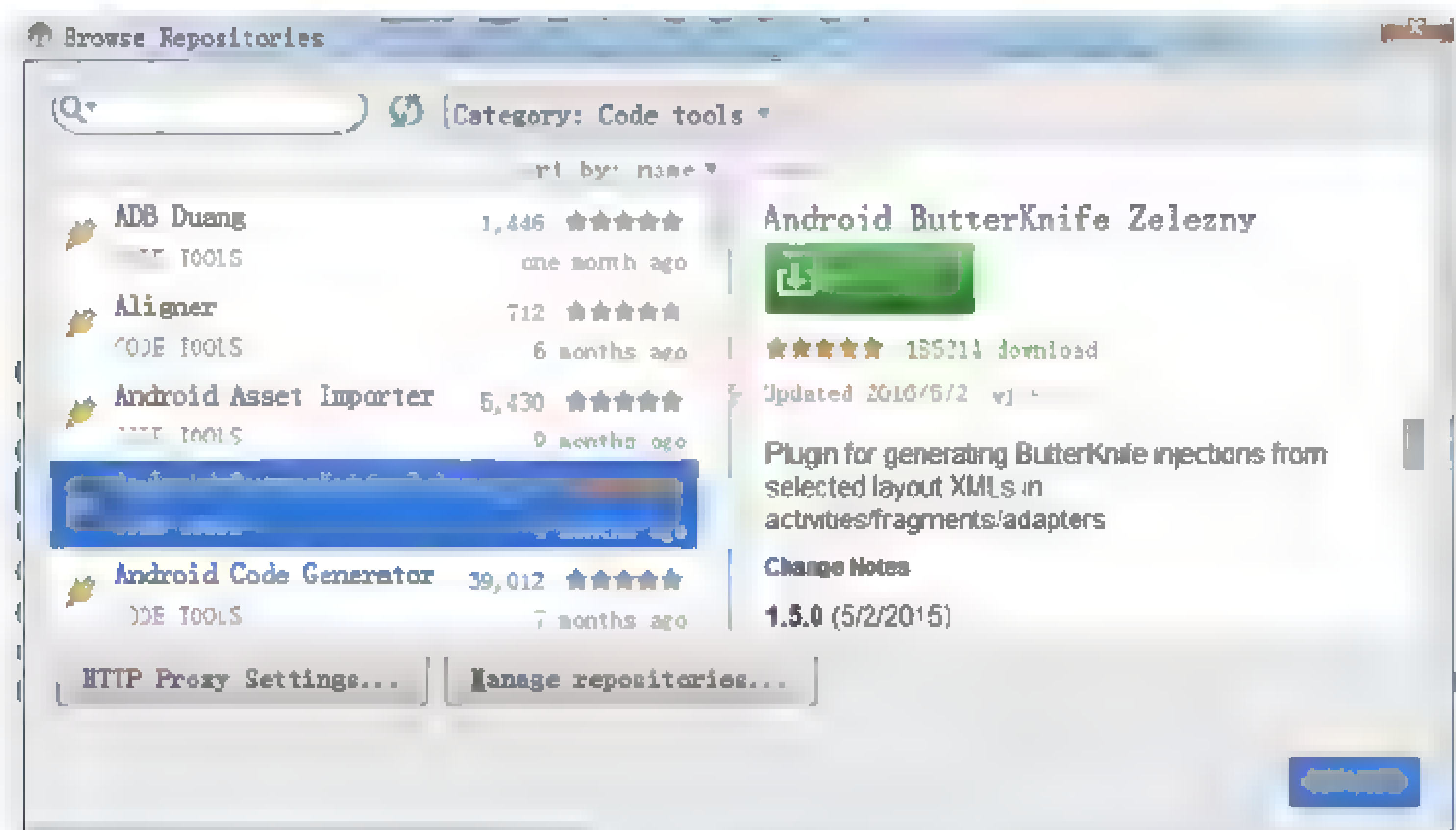


图 1-30 安装插件窗口

在安装插件窗口的 Category 框中选择 Code tools，然后选中左边列表的指定插件，再单击右边窗口内部的 Install 按钮，安装后重启 Studio 即可正常使用该插件的功能。下面是 5 个常用的 Studio 插件：

1. Android Parcelable code generator

该插件可自动生成 Parcelable 接口的代码。开发者先写好一个类和内部变量的定义，然后在代码中按 Alt+Insert，弹出的菜单列表下方就有 Parcelable 选项，如图 1-31 所示。选中该选项，即在类中插入实现 Parcelable 接口的代码。

2. Android Code Generator

该插件可根据布局文件快速生成对应的 Activity、Fragment、Adapter、Menu 等代码。在布局文件上右击或者在布局文件内部右击，弹出的菜单中多了一个 Generate Android Code 选项，具体的菜单如图 1-32 所示。选中生成项后，便会弹出代码窗口，把已生成的代码复制出来即可。注意该插件对汉字的支持不太好，如果 xml 文件中有汉字，代码就会生成失败。

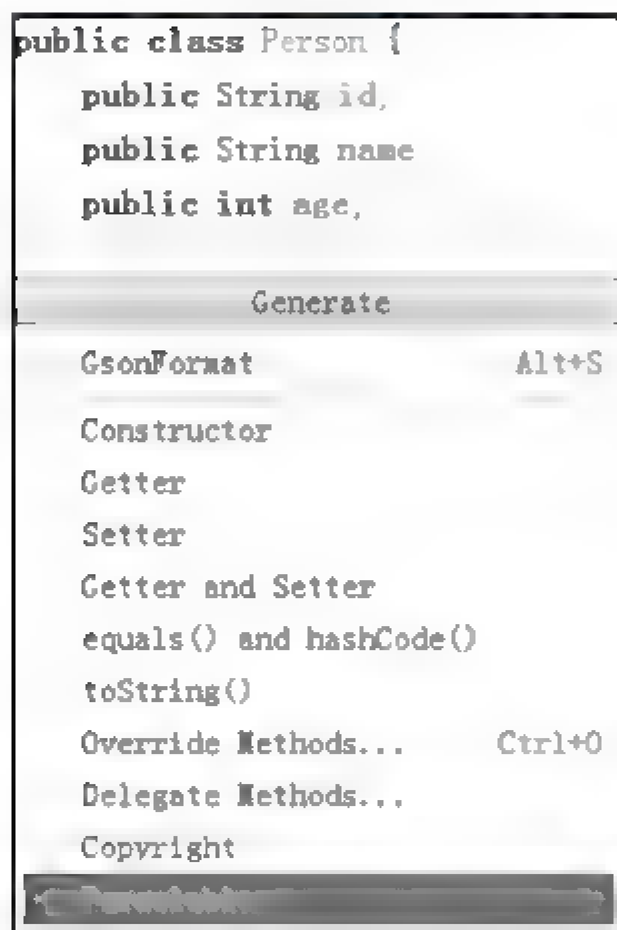


图 1-31 Parcelable 插件

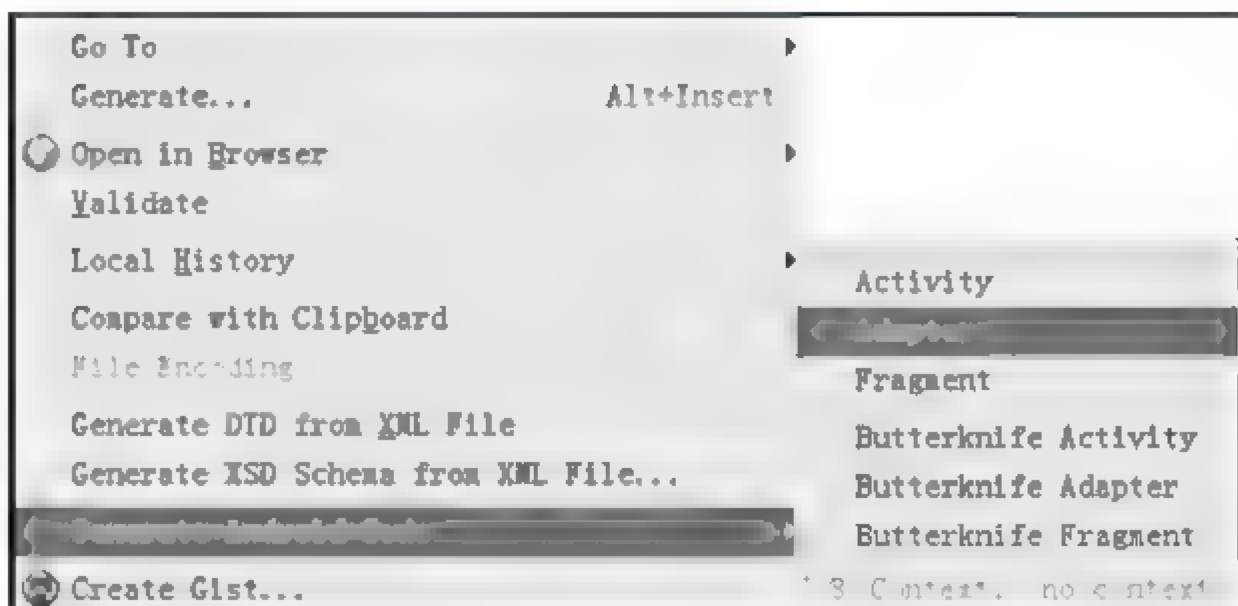


图 1-32 Generate Android Code 插件菜单

3. GsonFormat

该插件能够快速将 json 字符串转换成代码段，包含变量定义以及 set、get 函数。在代码中按 Alt+S，弹出 json 格式化窗口，往窗口中粘贴 json 字符串，单击 OK 按钮，即可在代码中插入生成好的代码段。GsonFormat 窗口如图 1-33 所示。



图 1-33 GsonFormat 插件

4. Android Postfix Completion

该插件支持在代码中快速生成 Toast、Log 等代码行。开发者在代码中输入字符串，后面



跟上.toast 并回车，即可生成 Toast.makeText 代码行；输入字符串后，紧接着输入.log 并回车，即可生成 Log.d 代码行，如图 1-34 所示。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    TextView tv_hello = (TextView) findViewById(R.id.tv_hello);
    //给TextView控件设置文字内容
    tv_hello.setText("今天天气真热啊，火辣辣的");

    tv_hello.setTextColor(Color.RED);

    tv_hel

    "测试日志".log
}
```

图 1-34　Postfix 插件使用截图

5. Android Drawable Importer

该插件可对一张图片自动生成不同分辨率的图片，从而让图片对不同屏幕的适配工作变得更加容易。右击任意目录，在弹出的菜单中选择 New，右方弹出的菜单列表末尾会出现*** Drawable Importer 之类的菜单项，如图 1-35 所示。

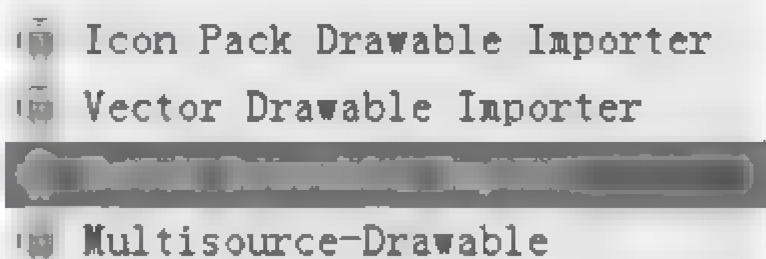


图 1-35　Drawable 插件菜单

这里通常选中 Batch Drawable Import，在弹出的窗口中选择图片的文件路径，并勾选需要自动生成的分辨率，然后单击 OK 按钮，即可在 drawabe 各分辨率的目录下生成对应的图片。

1.5.4　导入 ADT 工程

虽然现在 Android Studio 是 App 开发的主流工具，但是之前有不少 App 是基于 ADT 开发的，网络上也有许多源码以 ADT 工程的形式提供，所以在开发过程中会经常把原有的 ADT 工程导入 Android Studio 环境。

导入 ADT 工程的操作步骤是：打开 Android Studio，依次选择菜单 File→New→Import Module，然后单击窗口右边的浏览按钮，选择 ADT 工程的路径，单击 Finish 按钮，等待 Android Studio 识别并导入 ADT 工程。如果导入成功，接下来就能按照正常操作步骤编译和运行该工程的 App 了。

导入的 ADT 工程如果在运行时提示“Error:(1, 1) 错误：非法字符：'\ufeff'”，是因为源代码文件是带 BOM 的 utf8 格式，如果是 Eclipse 就会自动将它识别为正常的格式，但 Android Studio 目前还不会正常识别，所以要先把这种文件转换为无 BOM 的 utf8 格式。办法是打开 UEStudio 这类文本编辑软件，先把代码文件另存为无 BOM 的 utf8 格式文件，然后在 Android Studio 中刷新文件并重新编译。

1.6 小　结

本章主要介绍了 App 开发环境 —— Android Studio 环境的搭建。Android Studio 作为一个集成开发环境，依赖于 3 个开发工具：JDK、SDK、NDK。从创建最简单的 HelloWorld 项目开始，依次介绍了项目创建、项目编译、模拟器创建、在模拟器上运行 App 这一连串开发流程。为了让读者有更理性的认识，又逐步讲解了 App 的工程目录结构、编译配置文件 build.gradle 的使用说明、App 运行配置文件 AndroidManifest.xml 的节点说明、如何在代码中简单操作控件等。最后对开发过程中的准备工作做了必要的说明，主要包括如何使用快捷键、如何使用 SVN 进行版本管理、如何安装和使用常见插件、如何把 ADT 工程导入 Android Studio。

通过本章的学习，读者应该获得了 Android Studio 的基本操作技能，能够使用自己搭建的 Android Studio 环境创建简单的 App 并在模拟器上运行，并具备进一步提高的学习基础。





初级控件

本章介绍 Android 屏幕显示初级视图的相关知识，主要包括屏幕显示基础、简单布局的用法、简单控件的用法、简单图形的用法。并且结合本章所学的知识，演示了一个实战项目“简单计算器”的设计与实现。

2.1 屏幕显示

本节从最基础的显示单元开始介绍，讲述了移动设备如何在屏幕上展现丰富多彩的界面。本节主要内容包括像素的几个常用单位、颜色的编码与使用、屏幕分辨率的获取等。

2.1.1 像素

老子曾说“天下难事必作于易，天下大事必作于细”，Android 开发也是如此。纵使 App 的界面千变万化、绚丽多姿，也都归因于数百万个像素的组合排列，就像万物皆由原子构成一般。像素看似简单，实际有大学问，如果对像素单位不知其所以然，开发时只知一根筋的填数字，结果在模拟器上运行得很好的界面，在真机上很可能显示得东倒西歪，这就是没打好基础的缘故。如果一开始就把像素的基本概念弄清楚，后面就会少走很多弯路，开发起来也会更加得心应手。

Android 支持的像素单位有：px（像素）、in（英寸）、mm（毫米）、pt（磅，1/72 英寸）、dp（与设备无关的显示单位）、dip（就是 dp）、sp（用于设置字体大小）。其中，常用的有 px、dp 和 sp 三种。

具体来说，px 是手机屏幕上可显示的最小单位，与物理设备的显示屏有关。一般来说，同样尺寸的屏幕（比如 5 寸的手机）看起来越清晰，像素的密度越高，以 px 计量的分辨率也越大。

dp 与物理设备无关，只与屏幕的尺寸有关。一般来说，同样尺寸的屏幕以 dp 计量的分辨率是一样的，无论这个手机是哪个厂家生产的，dp 大小都一样。

sp 的原理跟 dp 差不多，专门用于设置字体大小。手机在系统设置里可以调整字体的大小（小、普通、大、超大）。设置普通字体时，同数值 dp 和 sp 的文字看起来一样大；如果设置为大字体，用 dp 设置的文字没有变化，用 sp 设置的文字就变大了。例如，当系统设置普通字体时，18dp 与 18sp 的文字一样大，如图 2-1 所示；当系统设置大字体时，18dp 的文字大小不变，18sp 的文字却增大了，如图 2-2 所示。

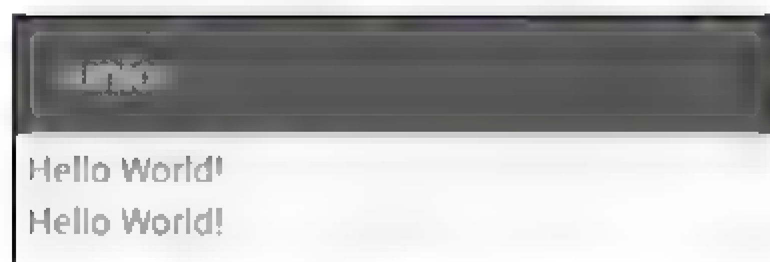


图 2-1　普通字体的效果图

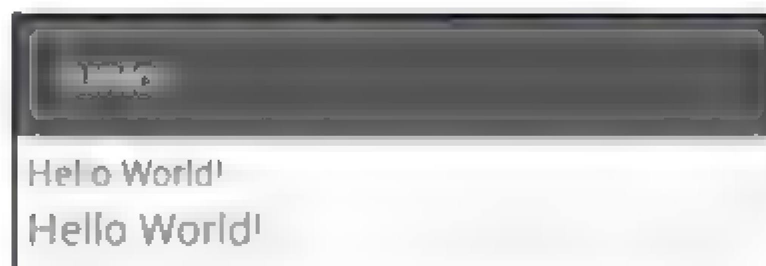


图 2-2　大字体的效果图

所以说，dp 与系统设置的字体大小没有关系，而 sp 会随系统设置的字体大小变大或变小。

dp 和 px 之间的联系取决于具体设备上的像素密度，像素密度就是 DisplayMetrics 里的 density 参数。当 density=1.0 时，表示一个 dp 值对应一个 px 值；当 density=1.5 时，表示两个 dp 值对应 3 个 px 值；当 density=2.0 时，表示一个 dp 值对应两个 px 值。具体的转换函数如下：

```
public class Utils {
    //根据手机的分辨率从 dp 的单位转成为 px(像素)
    public static int dip2px(Context context, float dpValue) {
```

```
            final float scale = context.getResources().getDisplayMetrics().density;
            return (int) (dpValue * scale + 0.5f);
        }

        //根据手机的分辨率从 px(像素) 的单位 转成为 dp
        public static int px2dip(Context context, float pxValue) {
            final float scale = context.getResources().getDisplayMetrics().density;
            return (int) (pxValue / scale + 0.5f);
        }
    }
```

在 XML 布局文件中，为了让不同设备屏幕拥有统一的显示效果，除了 sp 用于设置文字大小外，其余要用大小的地方都用 dp。在代码中情况又有所不同，Android 用于设置大小的函数都以 px 为单位。无论是 LayoutParams 里的 width 和 height，还是 setMargins 和 setPadding，参数单位都是 px，要想在代码中使用 dp 设置布局大小或间距，得先把 dp 值转换成 px 值。代码示例如下：

```
        int dip_10 = Utils.dip2px(this, 10L);
        TextView tv_padding = (TextView) findViewById(R.id.tv_padding);
        tv_padding.setPadding(dip_10, dip_10, dip_10, dip_10);
```

2.1.2　颜色

在 Android 中，颜色值由透明度 alpha 和 RGB（红、绿、蓝）三原色定义，有八位十六进制数与六位十六进制数两种编码，例如八位编码 FFEEDDCC，FF 表示透明度，EE 表示红色的浓度，DD 表示绿色的浓度，CC 表示蓝色的浓度。透明度为 FF 表示完全不透明，为 00 表示完全透明。RGB 三色的数值越大颜色越浓也就越亮，数值越小颜色越暗。亮到极致就是白色，暗到极致就是黑色，这样记就不会搞混了。

六位十六进制编码有两种情况，在 XML 文件中默认不透明（透明度为 FF），在代码中默认透明（透明度为 00）。下面的代码分别给两个文本控件设置六位编码和八位编码的背景色。

```
        TextView tv_code_six = (TextView) findViewById(R.id.tv_code_six);
        tv_code_six.setBackgroundColor(0x00ff00);
        TextView tv_code_eight = (TextView) findViewById(R.id.tv_code_eight);
        tv_code_eight.setBackgroundColor(0xff00ff00);
```

从图 2-3 可以看到，代码使用六位编码看不到任何背景，使用八位编码能够看到正确的绿色背景。

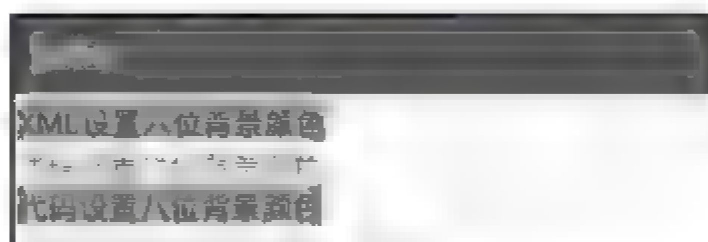


图 2-3　不同方式设置颜色编码的效果图

在 Android 中使用颜色有下列 3 种方式：

1. 使用系统已定义的颜色常量。

Android 系统有 12 种已经定义好的颜色，具体的类型定义在 Color 类中，详细的取值说明见表 2-1。

表 2-1 颜色类型的取值说明

Color 类中的颜色类型	说明	Color 类中的颜色类型	说明
BLACK	黑色	GREEN	绿色
DKGRAY	深灰	BLUE	蓝色
GRAY	灰色	YELLOW	黄色
LTGRAY	浅灰	CYAN	青色
WHITE	白色	MAGENTA	玫红
RED	红色	TRANSPARENT	透明

2. 使用十六进制的颜色编码。

在布局文件中设置颜色需要在色值前面加“#”，如 android:textColor="#000000"。在代码中设置颜色可以直接填八位的十六进制数值（如 setTextColor(0xff00ff00);），也可以通过 Color.rgb(int red, int green, int blue)和 Color.argb(int alpha, int red, int green, int blue)这两种方法指定颜色。在代码中一般不要用六位编码，因为六位编码在代码中默认透明，所以代码用六位编码跟不用没什么区别。

3. 使用 colors.xml 中定义的颜色。

res/values 目录下有个 colors.xml 文件，是颜色常量的定义文件。如果要在布局文件中使用 XML 颜色常量，可引用“@color/常量名”；如果要在代码中使用 XML 颜色常量，可通过这行代码获取：getResources().getColor(R.color.常量名)。

2.1.3 屏幕分辨率

在 App 编码中时常要取手机的屏幕分辨率（如当前屏幕的宽和高），然后动态调整界面上的布局。在代码中获取分辨率就是想办法获得 DisplayMetrics 对象，然后从该对象中获得宽度、高度、像素密度等信息。下面是 DisplayMetrics 类的常用属性说明。

- widthPixels：以 px 为单位计量的宽度值。
- heightPixels：以 px 为单位计量的高度值。
- density：像素密度，即一个 dp 单位包含多少个 px 单位。

下面是获取当前屏幕的宽度、高度、像素密度的代码示例。

```
public class DisplayUtil {
    public static int getSreenWidth(Context ctx) {
        WindowManager wm = (WindowManager) ctx.getSystemService(Context.WINDOW_SERVICE);
        DisplayMetrics dm = new DisplayMetrics();
        wm.getDefaultDisplay().getMetrics(dm);
```

```
            return dm.widthPixels;
        }

        public static int getSreenHeight(Context ctx) {
            WindowManager wm = (WindowManager) ctx.getSystemService(Context.WINDOW_SERVICE);
            DisplayMetrics dm = new DisplayMetrics();
            wm.getDefaultDisplay().getMetrics(dm);
            return dm.heightPixels;
        }

        public static float getSreenDensity(Context ctx) {
            WindowManager wm = (WindowManager) ctx.getSystemService(Context.WINDOW_SERVICE);
            DisplayMetrics dm = new DisplayMetrics();
            wm.getDefaultDisplay().getMetrics(dm);
            return dm.density;
        }
    }
```

从一个接入设备上获得屏幕分辨率信息，如图 2-4 所示。该设备为 5 寸屏幕，分辨率是 720*1280，像素密度是 2。



图 2-4　某手机上的分辨率信息

2.2　简单布局

本节开始介绍 Android 的基本视图和布局，首先说明基本视图 View 类的常用属性和方法，接着描述如何使用线性布局 LinearLayout，最后介绍滚动视图 ScrollView 的用法。

2.2.1　视图 View 的基本属性

View 是 Android 的基本视图，所有控件和布局都是由 View 类直接或间接派生而来的。故而 View 类的基本属性和方法是各控件和布局通用的，掌握好基本属性和方法，在哪里都能派上用场，能够举一反三、事半功倍。

下面是视图在 XML 布局文件中常用的属性定义说明。

- id：指定该视图的编号。
- layout_width：指定该视图的宽度。可以是具体的 dp 数值；可以是 match_parent，表示与上级视图一样宽；也可以是 wrap_content，表示与内部内容一样宽（内部内容若超过上级视图

的宽度，则该视图保持与上级视图一样宽，超出宽度的内容得进行滚动才能显示出来）。

- layout_height：指定该视图的高度。取值说明同 layout_width。
- layout_margin：指定该视图与周围视图之间的空白距离（包括上、下、左、右）。另有 layout_marginTop、layout_marginBottom、layout_marginLeft、layout_marginRight 分别表示单独指定视图与上边、下边、左边、右边视图的距离。
- minWidth：指定该视图的最小宽度。
- minHeight：指定该视图的最小高度。
- background：指定该视图的背景。背景可以是颜色，也可以是图片。
- layout_gravity：指定该视图与上级视图的对齐方式。对齐方式的取值说明见表 2-2，若同时适用多种对齐方式，则可使用竖线“|”把多种对齐方式拼接起来。

表 2-2　对齐方式的取值说明

XML 中的对齐方式	Gravity 类中的对齐方式	说明
left	LEFT	靠左对齐
right	RIGHT	靠右对齐
top	TOP	向上对齐
bottom	BOTTOM	向下对齐
center	CENTER	居中对齐
center_horizontal	CENTER_HORIZONTAL	水平方向居中
center_vertical	CENTER_VERTICAL	垂直方向居中

- padding：指定该视图边缘与内部内容之间的空白距离。另有 paddingTop、paddingBottom、paddingLeft、paddingRight 分别表示指定视图边缘与内容上边、下边、左边、右边的距离。
- visibility：指定该视图的可视类型。可视类型的取值说明见表 2-3。

表 2-3　可视类型的取值说明

XML 中的可视类型	View 类中的可视类型	说明
visible	VISIBLE	可见。默认值
invisible	INVISIBLE	不可见。虽然看不到但还占着位置
gone	GONE	消失。不仅看不到而且不占位置了

下面是视图在代码中常用的设置方法说明。

- setLayoutParams：设置该视图的布局参数。参数对象的构造函数可以设置视图的宽度和高度。其中，LayoutParams.MATCH_PARENT 表示与上级视图一样宽，也可以是 LayoutParams.WRAP_CONTENT，表示与内部内容一样宽；参数对象的 setMargins 方法可以设置该视图与周围视图之间的空白距离。
- setMinimumWidth：设置该视图的最小宽度。
- setMinimumHeight：设置该视图的最小高度。
- setBackgroundColor：设置该视图的背景颜色。
- setBackgroundDrawable：设置该视图的背景图片。



- setBackgroundResource：设置该视图的背景资源 id。
- setPadding：设置该视图边缘与内部内容之间的空白距离。
- setVisibility：设置该视图的可视类型。取值说明见表 2-3。

前面提到 margin 和 padding 两个概念，margin 是指当前视图与周围视图的距离，padding 是指当前视图与内部内容的距离。这么说可能有些抽象，所谓百闻不如一见，说得再多不如亲眼看看是怎么回事。我们来做一个实验，看看它们的显示效果有什么不同。下面是实验用的布局文件源代码，以背景色观察每个控件的区域范围：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:background="#00aaff"
    android:orientation="vertical"
    android:padding="5dp" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="20dp"
        android:background="#ffff99"
        android:padding="60dp" >

        <View
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="#ff0000" />
    </LinearLayout>
</LinearLayout>
```

最后的界面效果如图 2-5 所示。布局文件处于中间层的 LinearLayout，设置 margin 是 20dp、padding 是 60dp。从效果图可以看到，中间层与上级视图之间的距离大约是中间层与下级视图之间距离的三分之一，正好是 margin 和 padding 两个数值的比例。如此便从实际情况中印证了：layout_margin 指的是当前图层与外部图层的距离，而 padding 指的是当前图层与内部图层的距离。

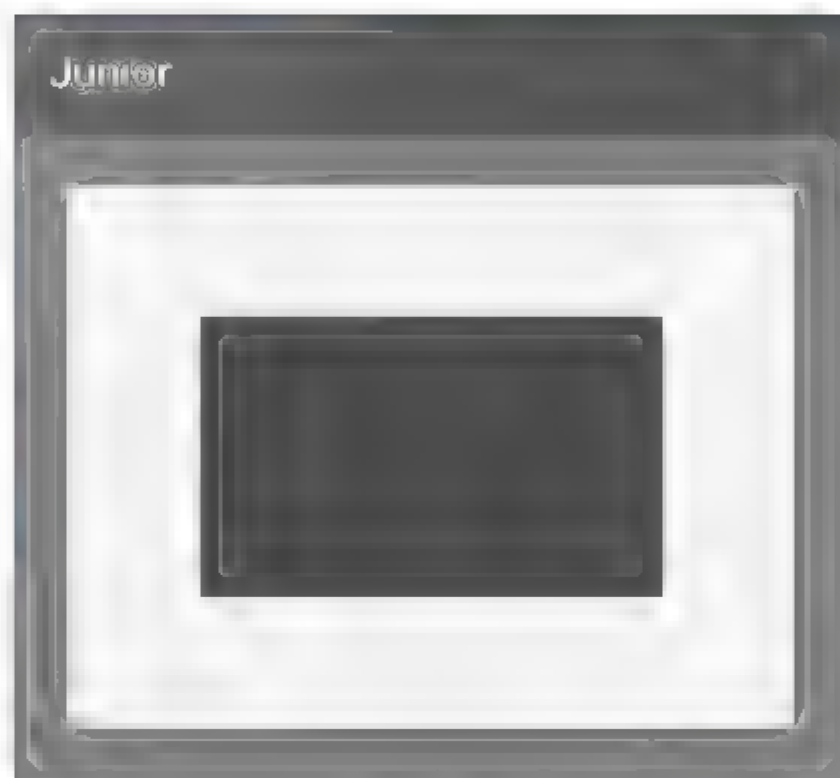


图 2-5 margin 和 padding 的演示画面

视图组 ViewGroup 是一类特殊视图，所有布局视图类都是从它派生而来的。Android 中的视图分为两类，一类是布局，另一类是控件。布局与控件的区别在于：布局本质上是个容器，里面还可以放其他视图（包括子布局和子控件）；控件是一个单

一的实体，已经是最后一级，下面不能再挂其他视图。打个比方，我们把根节点看作树干，根节点下的各级布局就是树枝，一根树枝可以连着其他小树枝，也可以直接连树叶；树叶只能依附在树枝上，不能再连树枝或其他树叶。

ViewGroup 有 3 个方法，这 3 个方法也是所有布局类视图共同拥有的。

- addView：往布局中添加一个视图。
- removeView：从布局中删除指定视图。
- removeAllViews：删除该布局下的所有视图。

2.2.2 线性布局 LinearLayout

LinearLayout 是最常用的布局，名字叫线性布局。顾名思义，LinearLayout 下面的子视图就像用一根线串了起来，所以 LinearLayout 内部视图的排列是有顺序的，要么从上到下依次垂直排列，要么从左到右依次水平排列。LinearLayout 除了继承 View/ViewGroup 类的所有属性和方法外，还有其特有的 XML 属性，说明如下。

- orientation：指定线性布局的方向。horizontal 表示水平布局，vertical 表示垂直布局。如果不指定该属性，就默认是 horizontal。这真是出乎意料，因为大家感觉手机 App 理应从上往下垂直布局，所以这里要特别注意垂直布局一定要设置 orientation，不然默认的水平布局不符合多数业务场景。
- gravity：指定布局内部视图与本线性布局的对齐方式。取值说明同 layout_gravity。
- layout_weight：指定当前视图的宽或高占上级线性布局的权重。这里要注意，layout_weight 属性并非在当前 LinearLayout 节点中设置，而是在下级视图的节点中设置。另外，如果 layout_weight 指定的是当前视图在宽度上占的权重，layout_width 就要同时设置为 0dp；如果 layout_weight 指定的是当前视图在高度上占的权重，layout_height 就要同时设置为 0dp。

下面是 LinearLayout 在代码中增加的两个方法。

- setOrientation：设置线性布局的方向。LinearLayout.HORIZONTAL 表示水平布局，LinearLayout.VERTICAL 表示垂直布局。
- setGravity：设置布局内部视图与本线性布局的对齐方式。具体的取值说明见表 2-2。

接下来重点解释 layout_gravity 和 gravity 的区别。前面说过，layout_gravity 指定该视图与上级视图的对齐方式，而 gravity 指定布局内部视图与本布局的对齐方式。为方便理解，我们通过一个具体例子演示两种属性的显示效果。下面是演示用的 XML 布局文件，内部指定了多种对齐方式，其中左边视图的 layout_gravity 是 bottom、gravity 是 left；右边视图的 layout_gravity 是 top、gravity 是 right，布局文件内容如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:background="#ffff99"
```

```
    android:orientation="horizontal"
    android:padding="5dp" >

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="200dp"
        android:layout_weight="1"
        android:layout_gravity="bottom"
        android:gravity="left"
        android:background="#ff0000"
        android:layout_margin="10dp"
        android:padding="10dp"
        android:orientation="vertical">

        <View
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:background="#00ffff" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="200dp"
        android:layout_weight="1"
        android:layout_gravity="top"
        android:gravity="right"
        android:background="#ff0000"
        android:layout_margin="10dp"
        android:padding="10dp"
        android:orientation="vertical">

        <View
            android:layout_width="100dp"
            android:layout_height="100dp"
            android:background="#00ffff" />
    </LinearLayout>
</LinearLayout>
```

运行后的界面效果如图 2-6 所示。从效果图可以看到，左边视图自身向下对齐，符合 layout_gravity 的设置，下级视图靠左对齐，符合 gravity 的设置；右边视图自身向上对齐，符合 layout_gravity 的设置，下级视图靠右对齐，符合 gravity 的设置。

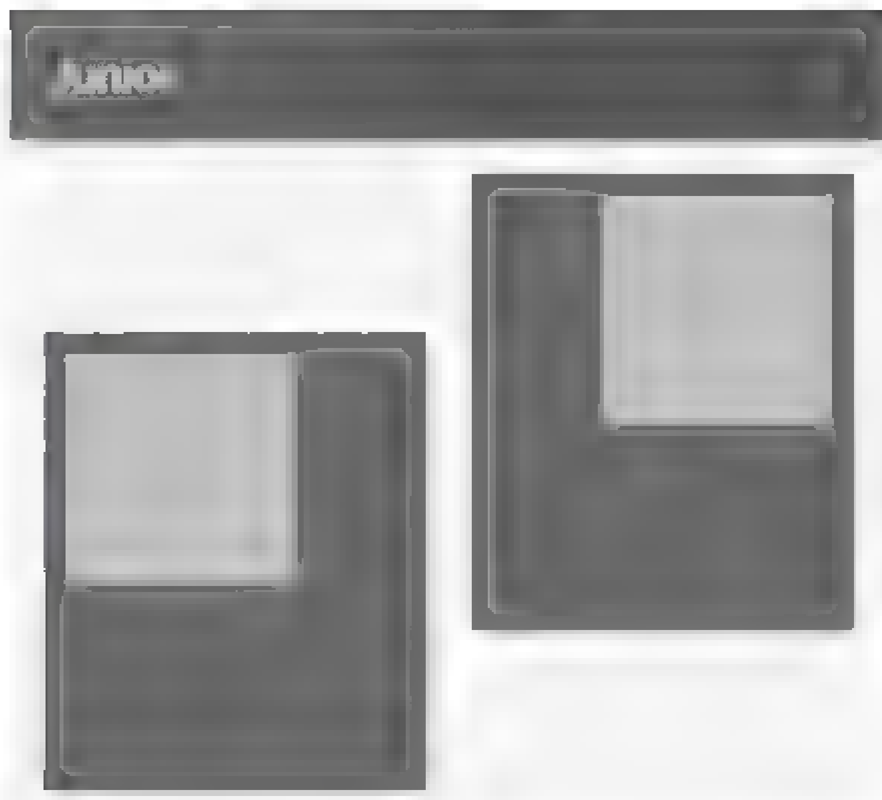


图 2-6　layout_gravity 和 gravity 的演示界面

2.2.3　滚动视图 ScrollView

手机屏幕的显示空间有限，常常需要上下滑动或左右滑动才能拉出其余页面内容，可惜 Android 的布局节点都不支持自行滚动，这时就要借助 ScrollView 滚动视图实现了。与线性布局类似，滚动视图也分为垂直方向和水平方向两类，其中垂直滚动的视图名是 ScrollView，水平滚动的视图名是 HorizontalScrollView。这两个滚动视图的使用并不复杂，主要注意以下 3 点：

（1）垂直方向滚动时，layout_width 要设置为 match_parent，layout_height 要设置为 wrap_content。

（2）水平方向滚动时，layout_width 要设置为 wrap_content，layout_height 要设置为 match_parent。

（3）滚动视图节点下面必须且只能挂着一个子布局节点，否则会在运行时报错 Caused by: java.lang.IllegalStateException：ScrollView can host only one direct child。

下面是滚动视图 ScrollView 和水平滚动视图 HorizontalScrollView 的 XML 用法示例：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <HorizontalScrollView
        android:layout_width="wrap_content"
        android:layout_height="200dp">

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:orientation="horizontal">

            <View
```

```
                    android:layout_width="400dp"
                    android:layout_height="match_parent"
                    android:background="#aaffff" />

                <View
                    android:layout_width="400dp"
                    android:layout_height="match_parent"
                    android:background="#ffff00" />
            </LinearLayout>
        </HorizontalScrollView>

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <View
                    android:layout_width="match_parent"
                    android:layout_height="400dp"
                    android:background="#00ff00" />

                <View
                    android:layout_width="match_parent"
                    android:layout_height="400dp"
                    android:background="#ffffaa" />
            </LinearLayout>
        </ScrollView>
    </LinearLayout>
```

有时 ScrollView 的实际内容不够，又想让它充满屏幕，怎么办呢？如果把 layout_height 属性赋值为 match_parent，那么结果还是不会充满，正确的做法是再增加一行 fillViewport 的属性设置，举例如下：

```
            android:layout_height="match_parent"
            android:fillViewport="true"
```

2.3 简单控件

本节介绍 Android 几个简单控件的用法与注意点，主要包括文本视图 TextView 的跑马灯与聊天室效果、按钮 Button 的监听器使用、图像视图 ImageView 的拉伸效果与截图功能、图像按钮 ImageButton 的适用场合等。

2.3.1 文本视图 TextView

TextView 是最基础的文本显示控件，常用的基本属性和设置方法见表 2-4。

表 2-4 TextView 的基本属性和设置方法说明

XML 中的属性	TextView类的设置方法	说明
text:	setText	设置文本内容
textColor	setTextColor	设置文本颜色
textSize	setTextSize	设置文本大小
textAppearance	setTextAppearance	设置文本风格，风格定义在 res/styles.xml
gravity	setGravity	设置文本的对齐方式，对应的方法是 setGravity。取值说明见表 2-2

读者对于这些基本属性和方法想必并不陌生，因为在第 1 章第一个 App “Hello World”中就用到了它们，这里不再赘述。接下来介绍 TextView 的两个特效用法。

1. 跑马灯效果

当一行文本的内容太多，导致无法全部显示，也不想分行展示时，只能让文字从左向右滚动显示，类似于跑马灯。电视在播报突发新闻时经常在屏幕下方轮播消息文字，比如“快讯：我国选手***在刚刚结束的**比赛中为中国代表团夺得第**枚金牌”。

跑马灯效果在 XML 布局文件中实现时需要额外指定部分属性，这些特殊属性及其设置方法的详细说明见表 2-5。

表 2-5 跑马灯用到的属性与方法说明

XML 中的属性	跑马灯用到的设置方法	说明
singleLine	setSingleLine	指定文本是否单行显示
ellipsize	setEllipsize	指定文本超出范围后的省略方式，省略方式的取值说明见表 2-6
focusable	setFocusable	指定是否获得焦点，跑马灯效果要求设置为 true
focusableInTouchMode	setFocusableInTouchMode	指定在触摸时是否获得焦点，跑马灯效果要求设置为 true

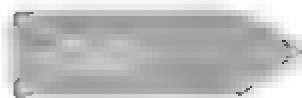


表 2-6　省略方式的取值说明

XML 中的省略方式	TruncateAt 类中省略方式	说明
start	START	省略号在开头
middle	MIDDLE	省略号在中间
end	END	省略号在末尾
marquee	MARQUEE	跑马灯显示

下面是演示跑马灯效果的 XML 布局文件：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:gravity="center"
        android:text="跑马灯效果，点击暂停，再点击恢复" />

    <TextView
        android:id="@+id/tv_marquee"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:singleLine="true"
        android:ellipsize="marquee"
        android:focusable="true"
        android:focusableInTouchMode="true"
        android:textColor="#000000"
        android:textSize="17sp"
        android:text="快讯 ： 红色预警，超强台风“莫兰蒂”即将登陆，请居民关紧门窗、备足粮
草，做好防汛救灾准备！" />
</LinearLayout>
```

跑马灯滚动的效果界面如图 2-7 和图 2-8 所示。左图为跑马灯文字在滚动中，右图为跑马灯文字停止滚动。

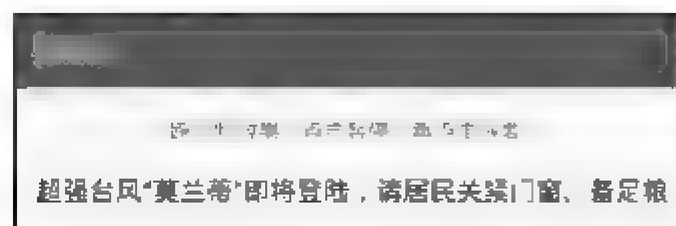


图 2-7　跑马灯文字滚动界面



图 2-8　跑马灯文字停止滚动界面

2. 聊天室或者文字直播间效果

聊天室窗口的高度是固定的，新的文字消息总是加入窗口末尾，同时窗口内部的文本整体向上滚动，窗口的大小、位置保持不变。

在 XML 布局文件中实现聊天室时需要额外指定部分属性，这些特殊属性及其设置方法的详细说明见表 2-7。

表 12-7　聊天室用到的属性与方法说明

XML 中的属性	聊天室用到的设置方法	说明
gravity	setGravity	指定文本的对齐方式，取值 left|bottom，表示靠左对齐且靠下对齐
lines	setLines	指定文本的行数
maxLines	setMaxLines	指定文本的最大行数
scrollbars	无	指定滚动条的方向，取值 vertical，如果不指定将不显示滚动条
无	setMovementMethod	设置文本的移动方式，可设置 ScrollingMovementMethod，如果不设置将无法拉动文本

接下来看一个简单聊天室的例子，点击聊天室窗口可以添加一条聊天记录，长按聊天室窗口可以清除所有聊天记录。聊天室的演示界面如图 2-9 和图 2-10 所示，图 2-10 比图 2-9 多添加了 3 条聊天记录，整个聊天记录的文字自动往上滚动。

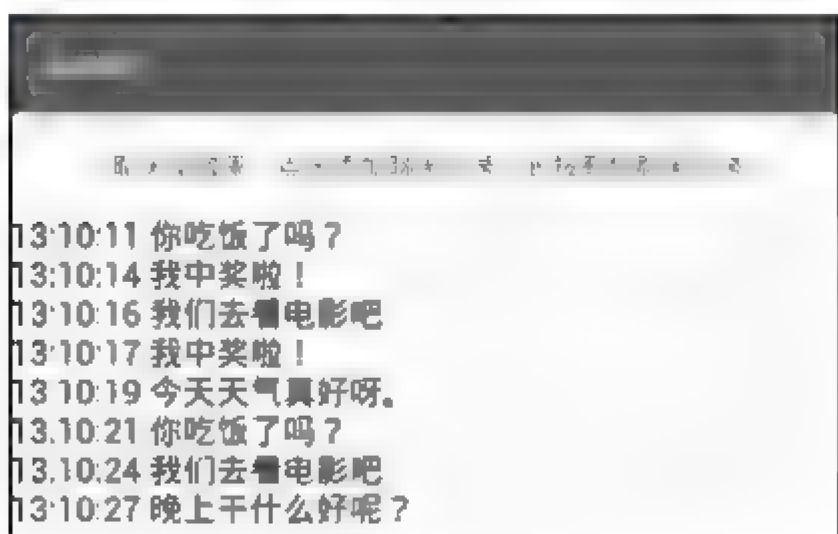


图 2-9　初始的聊天室界面

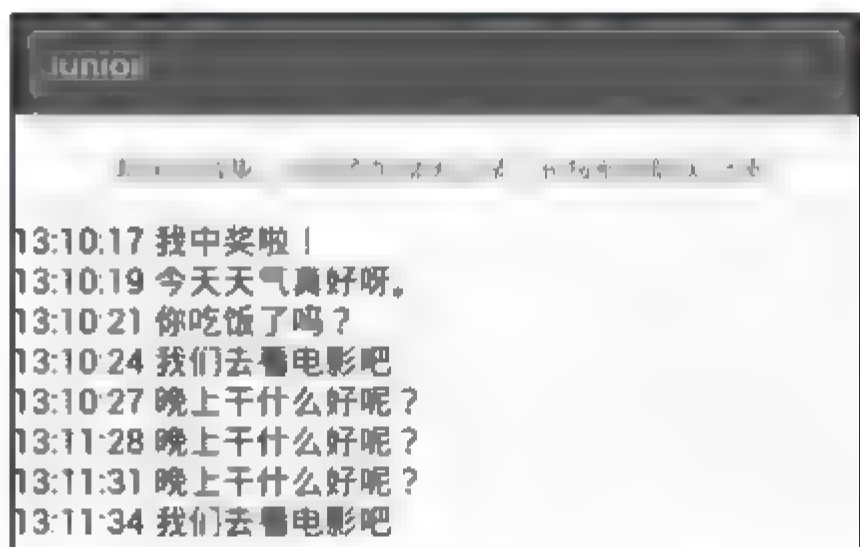


图 2-10　增加了 3 条聊天记录

下面是聊天室例子用到的 XML 布局文件内容：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/tv_control"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:gravity="center"
        android:text="聊天室效果，点击添加聊天记录，长按删除聊天记录" />
```



```
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="200dp"
            android:orientation="vertical">

            <TextView
                android:id="@+id/tv_bbs"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_marginTop="20dp"
                android:scrollbars="vertical"
                android:textColor="#000000"
                android:textSize="17sp" />
        </LinearLayout>
    </LinearLayout>
```

下面是聊天室例子用到的代码示例：

```
public class BbsActivity extends AppCompatActivity implements OnClickListener, OnLongClickListener {
    private TextView tv_bbs;
    private TextView tv_control;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_bbs);
        tv_control = (TextView) findViewById(R.id.tv_control);
        tv_control.setOnClickListener(this);
        tv_control.setOnLongClickListener(this);
        tv_bbs = (TextView) findViewById(R.id.tv_bbs);
        tv_bbs.setOnClickListener(this);
        tv_bbs.setOnLongClickListener(this);
        tv_bbs.setGravity(Gravity.LEFT|Gravity.BOTTOM);
        tv_bbs.setLines(8);
        tv_bbs.setMaxLines(8);
        tv_bbs.setMovementMethod(new ScrollingMovementMethod());
    }

    private String[] mChatStr = { "你吃饭了吗？", "今天天气真好呀。",
            "我中奖啦！", "我们去看电影吧", "晚上干什么好呢？", };
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.tv_control || v.getId() == R.id.tv_bbs) {
            int random = (int)(Math.random()*10) % 5;
            String newStr = String.format("%s\n%s %s",
                    tv_bbs.getText().toString(), DateUtil.getNowTime(), mChatStr[random]);
```

```
                tv_bbs.setText(newStr);
            }
        }

        @Override
        public boolean onLongClick(View v) {
            if (v.getId() == R.id.tv_control || v.getId() == R.id.tv_bbs) {
                tv_bbs.setText("");
            }
            return true;
        }
}
```

2.3.2　按钮 Button

Button 派生自 TextView，二者在 UI 上的区别主要是 Button 控件有个按钮外观，提示用户点击这里。系统默认的按钮外观通常都不好看，需要更换靓一点、活泼一点的图片，这时在布局文件中修改Button 节点的background 属性就可以了。如果把background 属性设置为@null，就会去除 Button 控件的背景样式，此时的 Button 看起来跟 TextView 没什么区别。

前面在演示聊天室功能时，我们为 TextView 引入了点击方法和长按方法。因为点击和长按监听器都来源于 View 类，所以这两个方法及其监听器并非 Button 特有的，而是所有布局和控件都能使用的，一般用于为按钮控件注册点击和长按事件。

Android 中的简单按钮主要是 Button 和后面提到的 ImageButton。这两个按钮对点击和长按监听器的使用方法并不复杂，主要步骤如下：

01 自己定义一个扩展自监听器的类，如点击监听器扩展自 View.OnClickListener，长按监听器扩展自 View.OnLongClickListener。为了方便起见，也可以直接给页面的 Activity 类加上监听器接口。

02 在自定义监听器类中重写点击或者长按方法，加入事件处理的代码。点击方法的名称是 onClick，长按方法的名称是 onLongClick。

03 哪个视图要响应点击或长按，就给哪个视图注册对应的监听器对象。点击事件的注册方法是 setOnClickListener，长按事件的注册方法是 setOnLongClickListener。

下面是给 Button 对象注册点击监听器和长按监听器的代码：

```
public class ClickActivity extends AppCompatActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_click);
            Button btn_click = (Button) findViewById(R.id.btn_click);
            btn_click.setOnClickListener(new MyOnClickListener());
            btn_click.setOnLongClickListener(new MyOnLongClickListener());
        }

        class MyOnClickListener implements View.OnClickListener {
```



```
			@Override
			public void onClick(View v) {
				if (v.getId() == R.id.btn_click) {
					Toast.makeText(ClickActivity.this, "您点击了控件 ： " + ((TextView) v).getText(),
Toast.LENGTH_SHORT).show();
				}
			}
		}

		class MyOnLongClickListener implements View.OnLongClickListener {
			@Override
			public boolean onLongClick(View v) {
				if (v.getId() == R.id.btn_click) {
					Toast.makeText(ClickActivity.this, "您长按了控件 ： " + ((TextView) v).getText(),
Toast.LENGTH_SHORT).show();
				}
				return true;
			}
		}
	}
```

2.3.3 图像视图 ImageView

ImageView 是图像显示控件，与图形显示有关的属性说明如下。

- scaleType：指定图形的拉伸类型，默认是 fitCenter。拉伸类型的取值说明见表 2-8。
- src：指定图形来源，src 图形按照 scaleType 拉伸。注意背景图不按 scaleType 指定的方式拉伸，背景默认以 fitXY 方式拉伸。

表 2-8 拉伸类型的取值说明

XML 中的拉伸类型	ScaleType 类中的拉伸类型	说明
fitXY	FIT_XY	拉伸图片使其正好填满视图（图片可能被拉伸变形）
fitStart	FIT_START	拉伸图片使其位于视图上部
fitCenter	FIT_CENTER	拉伸图片使其位于视图中间
fitEnd	FIT_END	拉伸图片使其位于视图下部
center	CENTER	保持图片原尺寸，并使其位于视图中间
centerCrop	CENTER_CROP	拉伸图片使其充满视图，并位于视图中间
centerInside	CENTER_INSIDE	使图片位于视图中间（只压不拉）。当图片尺寸大于视图时，centerInside 等同于 fitCenter；当图片尺寸小于视图时，centerInside 等同于 center

ImageView 在代码中调用的方法说明如下。

- setScaleType：设置图形的拉伸类型。具体的取值说明见表 2-8。
- setImageDrawable：设置图形的 Drawable 对象。



- setImageResource：设置图形的资源 ID。
- setImageBitmap：设置图形的位图对象。

读者应该注意到 ImageView 的拉伸类型种类繁多、文字说明不易理解，特别是 center 相关的类型就有 4 种：fitCenter、center、centerCrop、centerInside。接下来进行一个实验，把一张图片放入 ImageView 控件，尝试使用不同的拉伸类型，看看效果有什么区别。下面是图片拉伸演示用的代码示例：

```
public class ScaleActivity extends AppCompatActivity implements OnClickListener {
    private ImageView iv_scale;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_scale);
        iv_scale = (ImageView) findViewById(R.id.iv_scale);
        findViewById(R.id.btn_center).setOnClickListener(this);
        findViewById(R.id.btn_fitCenter).setOnClickListener(this);
        findViewById(R.id.btn_centerCrop).setOnClickListener(this);
        findViewById(R.id.btn_centerInside).setOnClickListener(this);
        findViewById(R.id.btn_fitXY).setOnClickListener(this);
        findViewById(R.id.btn_fitStart).setOnClickListener(this);
        findViewById(R.id.btn_fitEnd).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_center) {
            iv_scale.setScaleType(ImageView.ScaleType.CENTER);
        } else if (v.getId() == R.id.btn_fitCenter) {
            iv_scale.setScaleType(ImageView.ScaleType.FIT_CENTER);
        } else if (v.getId() == R.id.btn_centerCrop) {
            iv_scale.setScaleType(ImageView.ScaleType.CENTER_CROP);
        } else if (v.getId() == R.id.btn_centerInside) {
            iv_scale.setScaleType(ImageView.ScaleType.CENTER_INSIDE);
        } else if (v.getId() == R.id.btn_fitXY) {
            iv_scale.setScaleType(ImageView.ScaleType.FIT_XY);
        } else if (v.getId() == R.id.btn_fitStart) {
            iv_scale.setScaleType(ImageView.ScaleType.FIT_START);
        } else if (v.getId() == R.id.btn_fitEnd) {
            iv_scale.setScaleType(ImageView.ScaleType.FIT_END);
        }
    }
}
```

至于图像拉伸的演示界面，fitCenter 的效果如图 2-11 所示，图片被拉伸但未超出控件范围；center 的效果如图 2-12 所示，图片没有拉伸；centerCrop 的效果如图 2-13 所示，图片被拉伸且已超出控件范围；centerInside 的效果如图 2-14 所示，图片没有被拉伸。

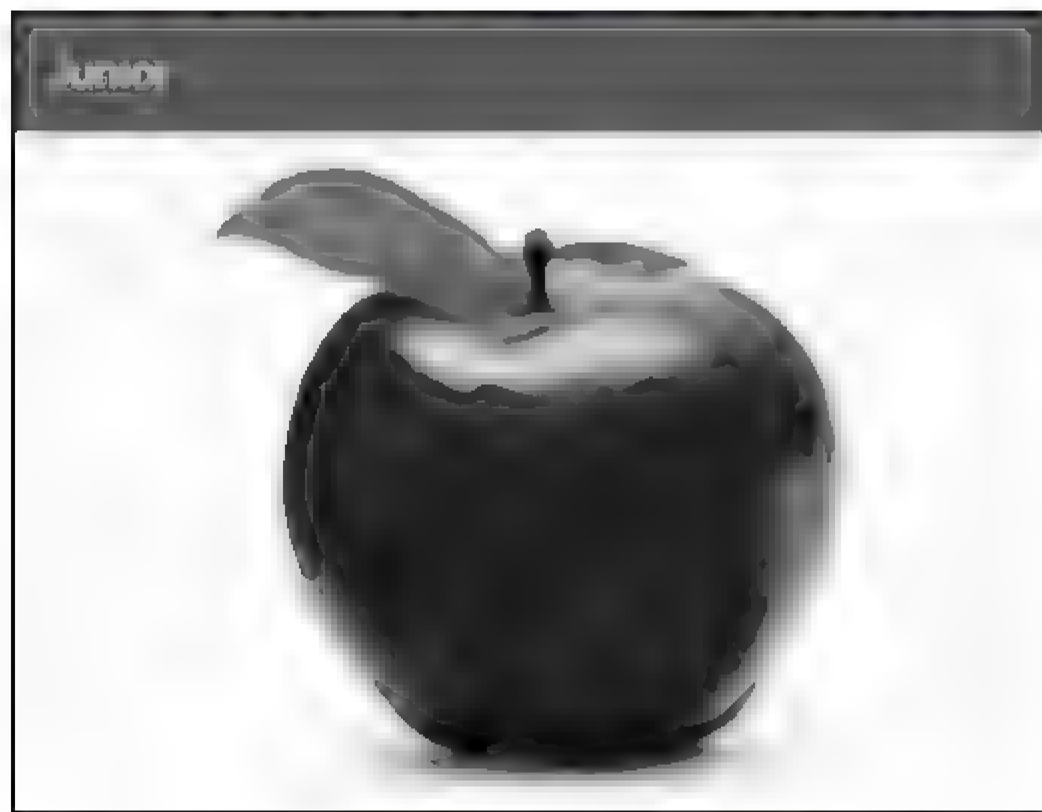


图 2-11　fitCenter 的效果图

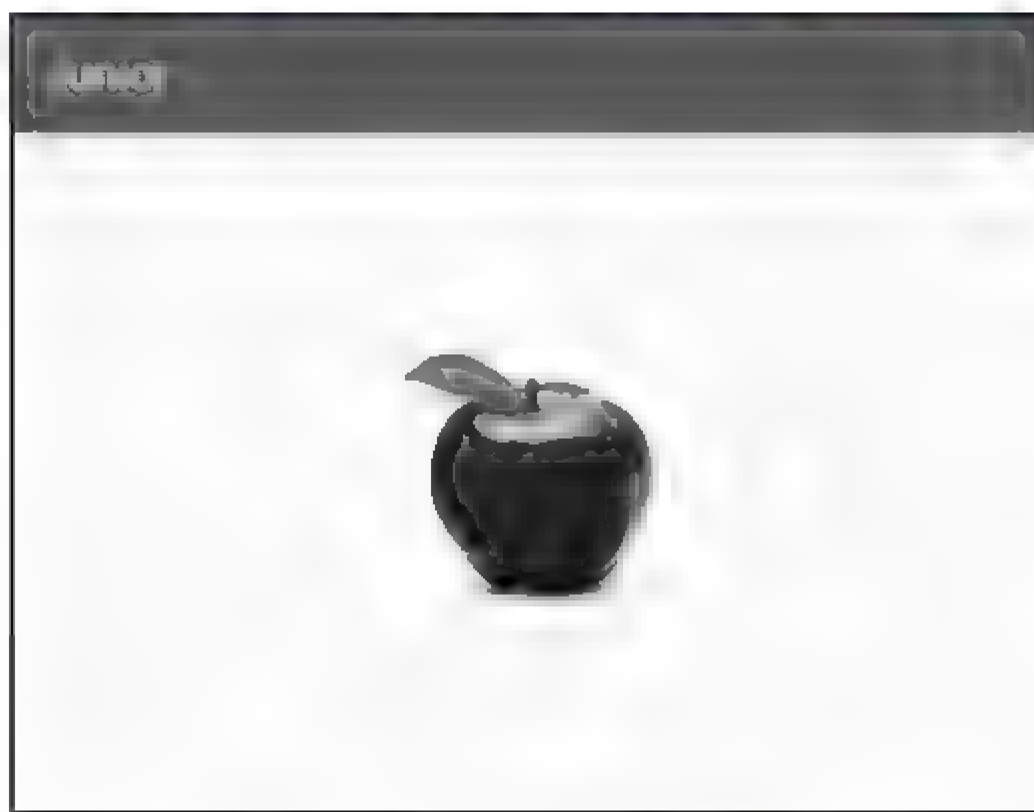


图 2-12　center 的效果图

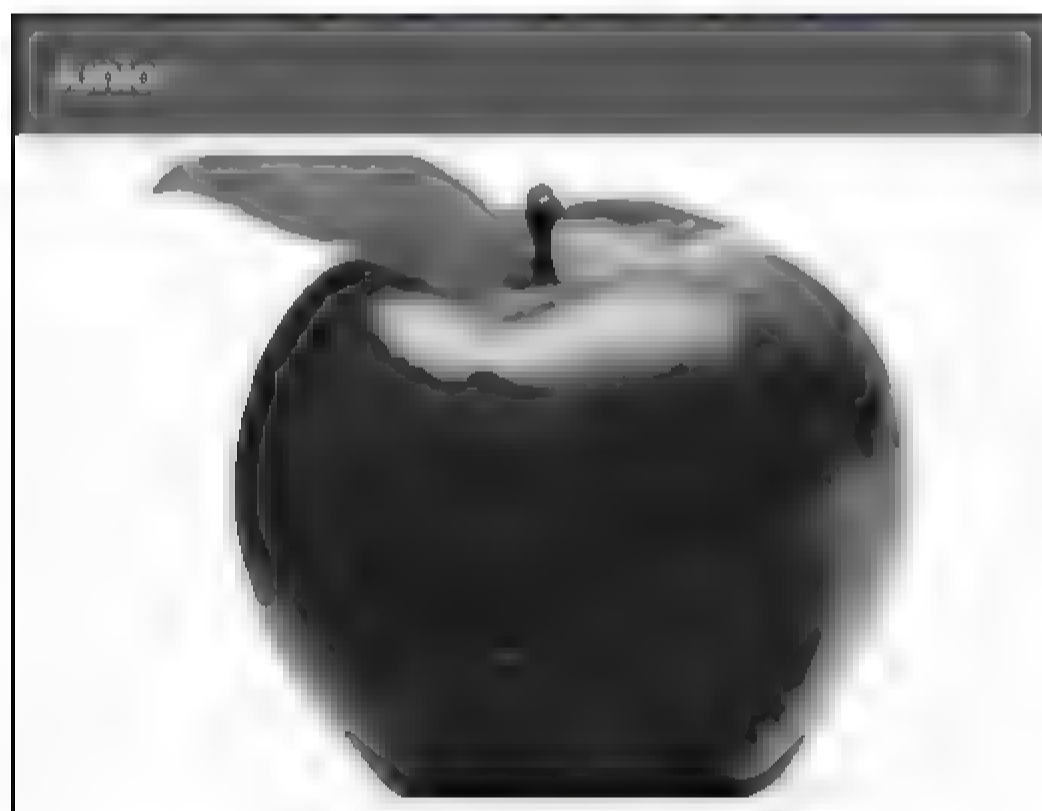


图 2-13　centerCrop 的效果图

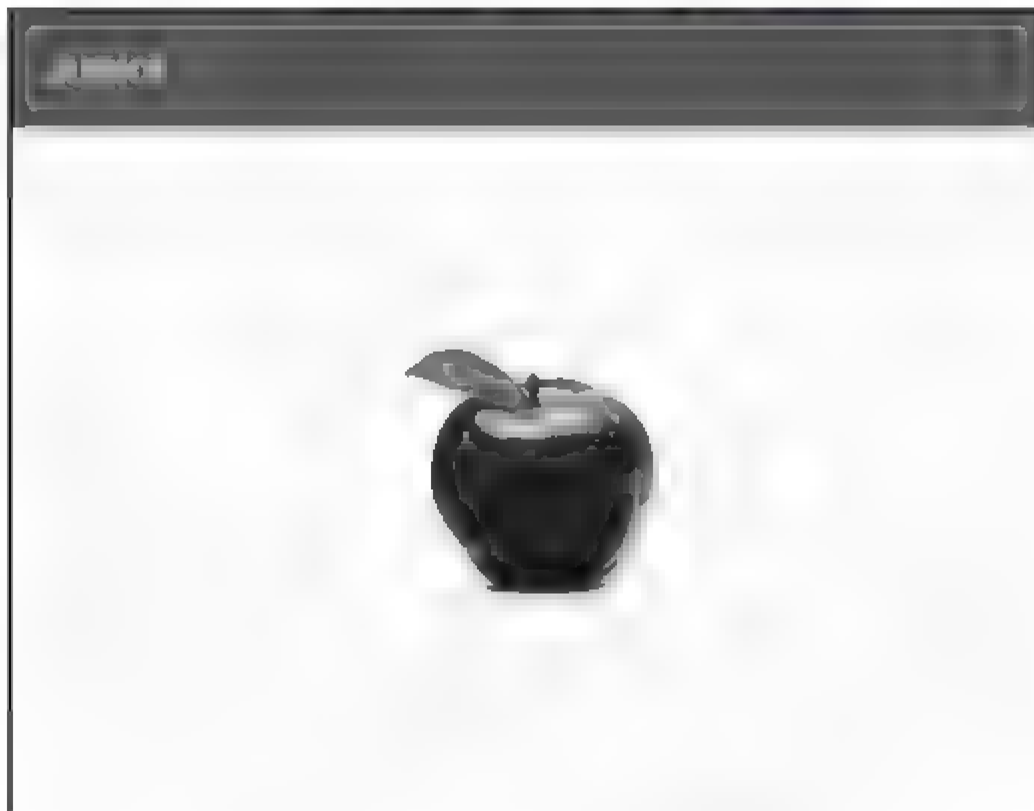


图 2-14　centerInside 的效果图

Android 能用 ImageView 展示图片，也自带屏幕截图功能。尽管自带的屏蔽截图功能有些简单，不过多数场合已经够用了。截图功能面向所有视图，我们可以从其他控件或布局那里截图下来，然后显示在 ImageView 上面。

使用截图功能必须通过代码完成，相关方法如下（这些方法都来自于 View 类）。

- setDrawingCacheEnabled：设置绘图缓存的可用状态。true 表示打开，false 表示关闭。
- isDrawingCacheEnabled：判断该控件的绘图缓存是否可用。
- setDrawingCacheQuality：设置绘图缓存的质量。
- getDrawingCache：获取该控件的绘图缓存结果，返回值为 Bitmap 类型。
- setDrawingCacheBackgroundColor：设置绘图缓存的背景颜色。大家可能会奇怪为何要提供该方法，因为绘图缓存默认背景色是黑色，如果不提前设置缓存的背景色，截图的结果就是黑乎乎一片，所以需要将背景色设置为默认颜色（通常是白色）。

操作截图功能的具体步骤如下：

步骤01 开始截图前，先调用 setDrawingCacheEnabled 方法，设置绘图缓存为可用状态。注意该方法在一开始就得调用，因为先开启绘图缓存，之后变更的界面才会记录到缓存中；如果先变更界面再开启绘图缓存，缓存里就是空的。

步骤02 调用 getDrawingCache 方法获取缓存中的图像数据。

步骤03 完成截图，延迟若干毫秒后调用 setDrawingCacheEnabled 方法关闭绘图缓存。如果接下来还要截图，就再次调用 setDrawingCacheEnabled 方法重新开启绘图缓存。

下面是完成截图功能的代码：

```
public class CaptureActivity extends AppCompatActivity implements OnClickListener, OnLongClickListener {
    private TextView tv_capture;
    private ImageView iv_capture;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_capture);
        tv_capture = (TextView) findViewById(R.id.tv_capture);
        iv_capture = (ImageView) findViewById(R.id.iv_capture);
        tv_capture.setDrawingCacheEnabled(true);
        findViewById(R.id.btn_chat).setOnClickListener(this);
        findViewById(R.id.btn_chat).setOnLongClickListener(this);
        findViewById(R.id.btn_capture).setOnClickListener(this);
    }

    private String[] mChatStr = { "你吃饭了吗？", "今天天气真好呀。",
            "我中奖啦！", "我们去看电影吧。", "晚上干什么好呢？" };
    @Override
    public boolean onLongClick(View v) {
        if (v.getId() == R.id.btn_chat) {
            tv_capture.setText("");
        }
        return true;
    }
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_chat) {
            int random = (int)(Math.random()*10) % 5;
            String newStr = String.format("%s\n%s %s",
                    tv_capture.getText().toString(), DateUtil.getNowTime(), mChatStr[random]);
            tv_capture.setText(newStr);
        } else if (v.getId() == R.id.btn_capture) {
```

```
                Bitmap bitmap = tv_capture.getDrawingCache();
                iv_capture.setImageBitmap(bitmap);
                // 注意截图完毕后不能马上关闭绘图缓存，因为界面渲染需要时间
                // 如果立即关闭缓存，渲染界面就会找不到位图对象，从而报错
                // java.lang.IllegalArgumentException：Cannot draw recycled bitmaps
                mHandler.postDelayed(mResetCache, 200);
            }
        }

        private Handler mHandler = new Handler();
        private Runnable mResetCache = new Runnable() {
            @Override
            public void run() {
                tv_capture.setDrawingCacheEnabled(false);
                tv_capture.setDrawingCacheEnabled(true);
            }
        };
    }
```

对应的截图演示界面如图 2-15 和图 2-16 所示。其中，图 2-15 所示为截图前的界面，图 2-16 所示为截图后的界面。

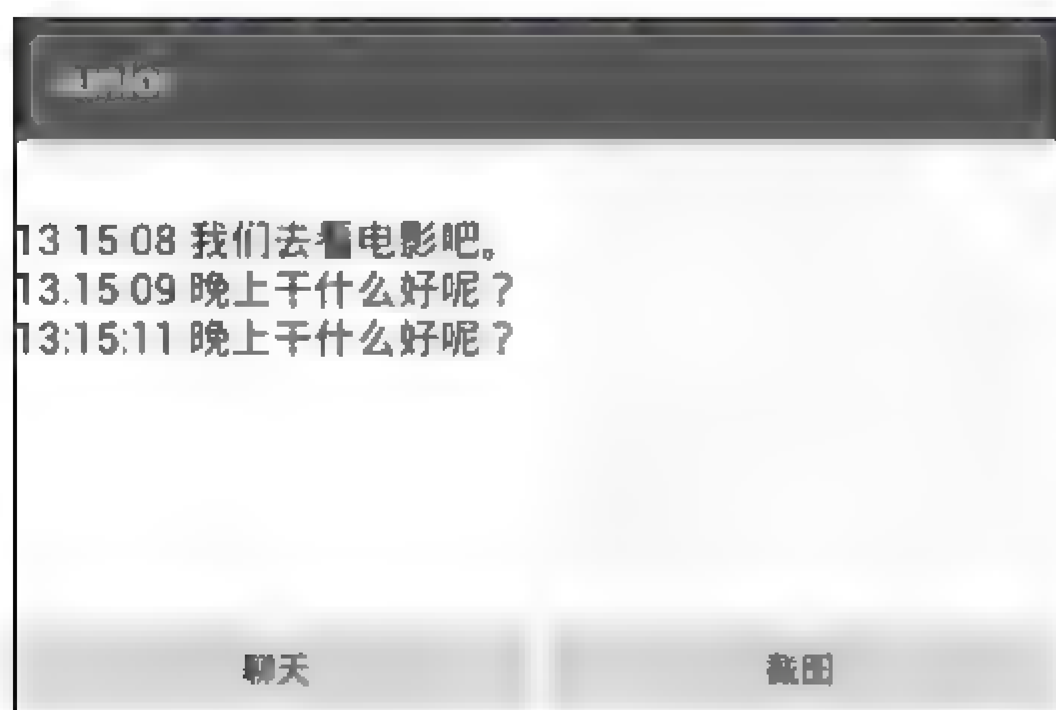


图 2-15　截图前只有左边有文字

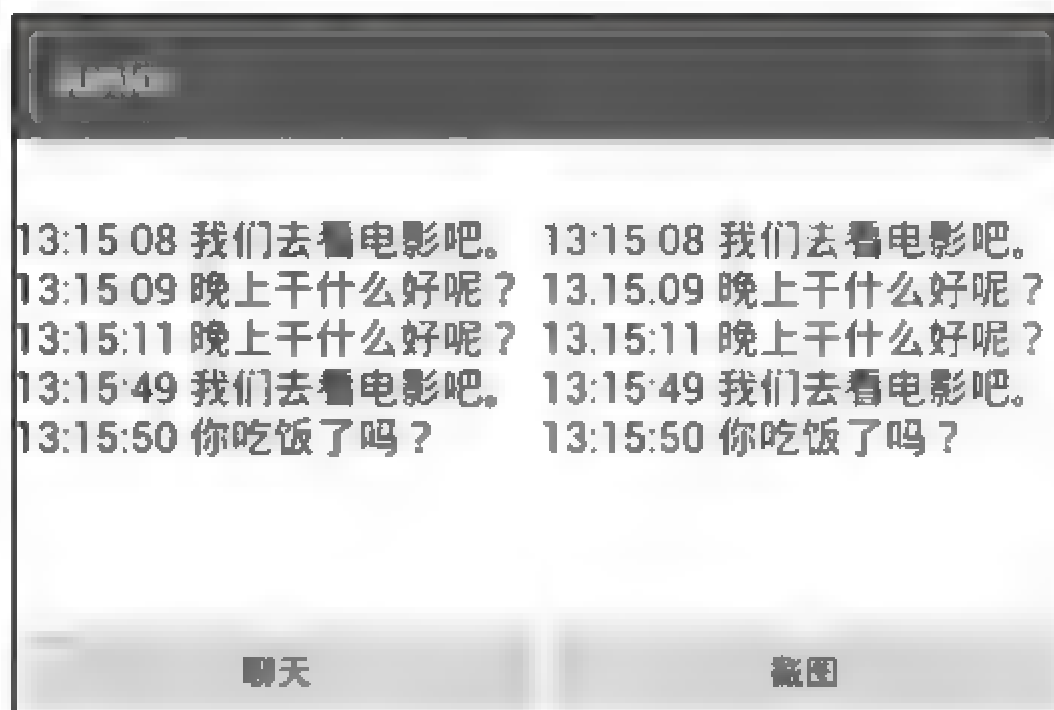


图 2-16　截图后在右边显示图片

2.3.4　图像按钮 ImageButton

ImageButton 其实派生自 ImageView，而不是派生自 Button，ImageView 拥有的属性和方法，ImageButton 统统拥有，只是 ImageButton 有个默认的按钮外观。

ImageButton 和 Button 都起到控制按钮的作用，不同的是 Button 是文本按钮，ImageButton 是图像按钮，这两个按钮的主要区别在于：

（1）Button 既可显示文本也可显示图形（通过设置背景图），而 ImageButton 只能显示图形不能显示文本。

（2）ImageButton 上的图像可按比例拉伸，而 Button 上的大图会拉伸变形（因为背景图无法按比例拉伸）。

（3）Button 只能在背景显示一张图形，而 ImageButton 可分别在前景和背景显示两张图形，实现图片叠加的效果。

从上面可以看出，Button 与 ImageButton 各有千秋，通常情况下使用 Button 就够用了。但在某些场合，比如输入法打不出来的字符和以特殊字体显示的字符串，就适合先切图再用 ImageButton 显示。

现在我们有了 Button 可在按钮上显示文字，又有 ImageButton 可在按钮上显示图形，照理说绝大多数场合都够用了。可是现实项目中的需求往往十分怪异，例如客户要求在按钮文字的左边加一个图标，这样按钮内部既有文字又有图片，乍看之下 Button 和 ImageButton 都没法直接使用。若把图标和文字放在一起切图，每次图标与文字的大小或距离发生变化时岂不是都要重新切图？若用 LinearLayout 对 ImageView 和 TextView 组合布局，这样固然可行，但是布局文件会冗长许多。

其实有个既简单又灵活的办法，要想在文字周围放置图片，使用 TextView 就能实现，那么基于 TextView 的 Button 自然能实现。具体可在 XML 布局文件中设置以下 5 个属性。

- drawableTop：指定文本上方的图形。
- drawableBottom：指定文本下方的图形。
- drawableLeft：指定文本左边的图形。
- drawableRight：指定文本右边的图形。
- drawablePadding：指定图形与文本的间距。

若在代码中实现，则可调用如下方法。

- setCompoundDrawables：设置文本周围的图形。可分别设置左边、上边、右边、下边的图形。
- setCompoundDrawablePadding：设置图形与文本的间距。

下面的代码演示在按钮中变换图标位置的功能：

```
public class IconActivity extends AppCompatActivity implements OnClickListener {
    private Button btn_icon;
    private Drawable drawable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_icon);
        btn_icon = (Button) findViewById(R.id.btn_icon);
        drawable = getResources().getDrawable(R.mipmap.ic_launcher);
        // 必须设置图片大小，否则不显示图片
        drawable.setBounds(0, 0, drawable.getMinimumWidth(), drawable.getMinimumHeight());
        findViewById(R.id.btn_left).setOnClickListener(this);
```

```
            findViewById(R.id.btn_top).setOnClickListener(this);
            findViewById(R.id.btn_right).setOnClickListener(this);
            findViewById(R.id.btn_bottom).setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.btn_left) {
                btn_icon.setCompoundDrawables(drawable, null, null, null);
            } else if (v.getId() == R.id.btn_top) {
                btn_icon.setCompoundDrawables(null, drawable, null, null);
            } else if (v.getId() == R.id.btn_right) {
                btn_icon.setCompoundDrawables(null, null, drawable, null);
            } else if (v.getId() == R.id.btn_bottom) {
                btn_icon.setCompoundDrawables(null, null, null, drawable);
            }
        }
    }
```

变换图标位置的效果界面如图 2-17（图标在文字左边）、图 2-18（图标在文字右边）、图 2-19（图标在文字上边）、图 2-20（图标在文字下边）所示。



图 2-17　图标在文字左边

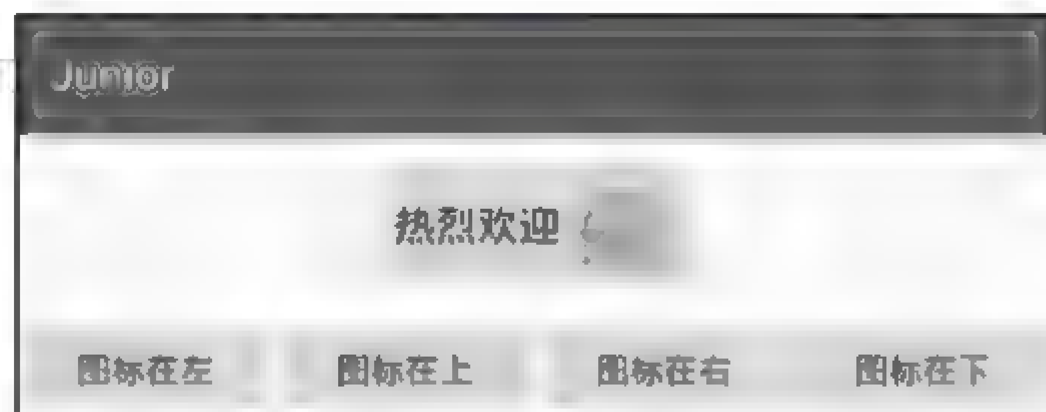


图 2-18　图标在文字右边



图 2-19　图标在文字上边



图 2-20　图标在文字下边

2.4　图形基础

本节介绍 Android 图形的基本概念和几种常见图形的使用方法，主要包括状态列表图形

StateListDrawable 的定义与使用、形状图形 ShapeDawable 的定义与使用、九宫格图片（点九图片）的制作与适用场景等。

2.4.1 图形 Drawable

Android 把所有显示出来的图形都抽象为 Drawable（可绘制的）。这里的图形不止是图片，还包括色块、画板、背景等。

drawable 文件放在 res 目录的各个 drawable 目录下。\res\drawable 一般存放的是描述性的 XML 文件，图片文件一般放在具体分辨率的 drawable 目录下。例如：

- drawable-ldpi 里面存放低分辨率的图片（如 240×320），现在基本没有这样的智能手机了。
- drawable-mdpi 里面存放中等分辨率的图片（如 320×480），这样的智能手机已经很少了。
- drawable-hdpi 里面存放高分辨率的图片（如 480×800），一般对应 4 寸~4.5 寸的手机（但不绝对，同尺寸的手机有可能分辨率不同，手机分辨率就高不就低，因为分辨率低了屏幕会有模糊的感觉）。
- drawable-xhdpi 里面存放加高分辨率的图片（如 720×1280），一般对应 5 寸~5.5 寸的手机。
- drawable-xxhdpi 里面存放超高分辨率的图片（如 1080×1920），一般对应 6 寸~6.5 寸的手机。
- drawable-xxxhdpi 里面存放超超高分辨率的图片（如 1440×2560），一般对应 7 寸以上的平板电脑。

基本上，分辨率每加大一级，宽度和高度就要加大二分之一或三分之一像素。如果各目录存在同名图片，Android 就会根据手机的分辨率分别适配对应文件夹里的图片。在开发 App 时，为了兼容不同的手机屏幕，根据需求在各目录存放不同分辨率的图片才能达到最合适的显示效果。例如，在 drawable-hdpi 放了一张背景图片 bg.png（分辨率 480×800），其他目录没放，使用分辨率 480×800 的手机查看该 App 没有问题，但是使用分辨率 720×1280 的手机查看 App 会发现背景图片有点模糊，原因是 Android 为了让 bg.png 适配高分辨率的屏幕，把 bg.png 拉伸到了 720×1280，拉伸的后果是图片变得模糊。

开发者拿到一张图片，可以直接复制粘贴到 drawable 目录，也可以通过批量 drawable 插件 Android Postfix Completion 生成并导入各分辨率的图片，该插件的安装和使用方法参见第 1 章的“1.5.3 安装常用插件”。

在 XML 布局文件中引用 drawable 文件可使用“@drawable/***”这种形式，如 background 属性、ImageView 和 ImageButton 的 src 属性、TextView 和 Button 的 drawableTop 系列属性都可以引用 drawable 文件。

在代码中引用 drawable 文件可分为两种情况：

（1）使用 setBackgroundResource 和 setImageResource 方法，可直接在参数中指定 drawable 文件的资源 ID，例如“R.drawable.***”。



（2）使用 setBackgroundDrawable、setImageDrawable 和 setCompoundDrawables 等方法，参数是 Drawable 对象，这时得先从资源文件中生成 Drawable 对象，示例代码如下：

```
Drawable drawable = getResources().getDrawable(R.drawable.apple);
```

2.4.2　状态列表图形

一般 drawable 是静态图形，如 Button 按钮的背景在正常情况下是凸起的，在按下时是凹陷的，从按下到弹起的过程，用户便能知道点击了这个按钮。根据不同的触摸情况变更图形显示，这种情况会用到 Drawable 的一个子类 StateListDrawable，该子类在 XML 文件中定义不同状态时呈现图形列表。

下面是一个状态列表图形的 drawable 文件：

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true" android:drawable="@drawable/button_pressed" />
    <item android:drawable="@drawable/button_normal" />
</selector>
```

该 XML 定义文件中的关键点是 state_pressed，值为 true 表示按下时显示 button_pressed 图像，其余情况显示 button_normal 图像。

为方便理解，接下来我们先将 Button 控件的 background 属性设置为该 drawable 文件，然后在屏幕上点击这个按钮，看看按下和弹起时分别呈现什么效果，界面如图 2-21（按下按钮）、图 2-22（按钮弹起）所示。



图 2-21　按下按钮时的背景样式



图 2-22　按钮弹起时的背景样式

StateListDrawable 不仅用于 Button 控件，而且可以用于其他拥有不同状态的控件，取决于开发者对 StateListDrawable 状态类型的定义。状态类型的取值说明见表 2-9。

表 2-9　状态类型的取值说明

状态类型	说明	常用的控件
state_pressed	是否按下	按钮 Button
state_checked	是否勾选	单选框 RadioButton、复选框 CheckBox
state_focused	是否获取焦点	文本编辑框 EditText
state_selected	是否选中	各控件均可

2.4.3 形状图形

前面讲到可在 XML 文件中描述状态列表图形的定义，还有一种常用的 XML 图形文件，是描述形状定义的图形—— shape 图形。用好 shape 可以让 App 页面不再呆板，还可以节省美工不少工作量。

形状图形的定义文件以 shape 元素为根节点。根节点下定义了 6 个节点：corners（圆角）、gradien（渐变）、padding（间隔）、size（尺寸）、solid（填充）、stroke（描边），各节点的属性值主要是长宽、半径、角度以及颜色。下面是形状图形各个节点和属性的简要说明。

1. shape

shape 是 XML 文件的根节点，用来描述该形状图形是哪种几何图形。下面是 shape 节点的常用属性说明。

- shape：字符串类型，图形的形状。形状类型的取值说明见表 2-10。

表 2-10 形状类型的取值说明

形状类型	说明
rectangle	矩形。默认值
oval	椭圆。此时 corners 节点会失效
line	直线。此时必须设置 stroke 节点，不然会报错
ring	圆环

2. corners

corners 是 shape 的下级节点，用来描述 4 个圆角的规格定义。若无 corners 节点，则表示没有圆角。下面是 corners 节点的常用属性说明。

- bottomLeftRadius：像素类型，左下圆角的半径。
- bottomRightRadius：像素类型，右下圆角的半径。
- topLeftRadius：像素类型，左上圆角的半径。
- topRightRadius：像素类型，右上圆角的半径。
- radius：像素类型，圆角半径（若有上面 4 个圆角半径的定义，则不需要 radius 定义）。

3. gradient

gradient 是 shape 的下级节点，用来描述形状内部的颜色渐变定义。若无 gradient 节点，则表示没有渐变效果。下面是 gradient 节点的常用属性说明。

- angle：整型，渐变的起始角度。为 0 时表示时钟的 9 点位置，值增大表示往逆时针方向旋转。例如，值为 90 表示 6 点位置，值为 180 表示 3 点位置，值为 270 表示 0 点/12 点位置。
- type：字符串类型，渐变类型。渐变类型的取值说明见表 2-11。

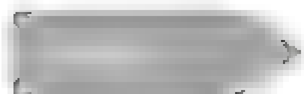


表 2-11　渐变类型的取值说明

渐变类型	说明
linear	线性渐变，默认值
radial	放射渐变，起始颜色就是圆心颜色
sweep	滚动渐变，即一个线段以某个端点为圆心做 360 度旋转

- centerX：浮点型，圆心的 X 坐标。当 android:type="linear"时不可用。
- centerY：浮点型，圆心的 Y 坐标。当 android:type="linear"时不可用。
- gradientRadius：整型，渐变的半径。当 android:type="radial"时才需要设置该属性。
- centerColor：颜色类型，渐变的中间颜色。
- startColor：颜色类型，渐变的起始颜色。
- endColor：颜色类型，渐变的终止颜色。
- useLevel：布尔类型，设置为 true 无渐变色、false 有渐变色。

4. padding

padding 是 shape 的下级节点，用来描述形状图形与周围视图的间隔大小。若无 padding 节点，则表示四周不设间隔。下面是 padding 节点的常用属性说明。

- bottom：像素类型，与下边的间隔。
- left：像素类型，与左边的间隔。
- right：像素类型，与右边的间隔。
- top：像素类型，与上边的间隔。

5. size

size 是 shape 的下级节点，用来描述形状图形的尺寸大小（宽度和高度）。若无 size 节点，则表示宽高自适应。下面是 size 节点的常用属性说明。

- height：像素类型，图形高度。
- width：像素类型，图形宽度。

6. solid

solid 是 shape 的下级节点，用来描述形状图形内部的填充色彩。若无 solid 节点，则表示无填充颜色。下面是 solid 节点的常用属性说明。

- color：颜色类型，内部填充的颜色。

7. stroke

stroke 是 shape 的下级节点，用来描述形状图形四周边线的规格定义。若无 stroke 节点，则表示不存在描边。下面是 stroke 节点的常用属性说明。

- color：颜色类型，描边的颜色。
- dashGap：像素类型，每段虚线之间的间隔。

- dashWidth：像素类型，每段虚线的宽度。
- width：像素类型，描边的厚度。若 dashGap 和 dashWidth 有一个值为 0，则描边为实线。

在实际开发中，常用的有 3 个节点：corners（圆角）、solid（填充）和 stroke（描边）。shape 根节点的属性一般不用设置（默认矩形就好了）。下面是 shape 图形的 XML 描述文件代码：

```
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
<solid android:color="#ffdd66" />
    <stroke
        android:width="1dp"
        android:color="#ffaaaaaa" />
    <corners
        android:bottomLeftRadius="10dp"
        android:bottomRightRadius="10dp"
        android:topLeftRadius="10dp"
        android:topRightRadius="10dp" />
</shape>
```

对应的形状图形效果界面如图 2-23 所示。该形状为一个圆角矩形，内部填充色为土黄色，边缘线为灰色。



图 2-23　shape 文件定义的圆角矩形效果

现在有个需求，客户要求在界面上增加一个水平分割线，如果是你会怎么做呢？按照目前为止的学习成果有以下 3 个办法。

（1）在 TextView 控件中连续填入许多横线或下划线。
（2）让美工做一个横线的切图，然后将 ImageView 控件塞进横线图。
（3）使用刚学的 shape，根节点的 shape 属性设置为 line 表示直线图形。

以上做法各有千秋，不过杀鸡焉用牛刀，简单的事情自然有简单的办法。最简单的做法是在布局文件中增加一个 View 控件，高度设置为 1dp、背景颜色设置为线条颜色，这样便实现了水平分割线的需求。XML 文件的示例代码如下：



```
<View
    android:layout_width="match_parent"
    android:layout_height="1dp"
    android:background="#000000" />
```

2.4.4　九宫格图片

前面在介绍 ImageView 时专门举了例子说明不同拉伸类型下的图片显示效果。当图片被拉大时，画面容易模糊，如果把图片作为背景图，模糊的情况会更严重。如图 2-24 所示，一张按钮图片被拉得很宽，此时左右两边的边缘线既变宽又变模糊了。

为了解决这个问题，Android 专门设计了点九图片。点九图片的扩展名是 png，文件名后常带有“.9”字样。因为把一张图片划分成了 3×3 的九宫格区域，所以得名点九图片，也叫九宫格图片。如果背景是一个 shape 图形，其 stroke 节点的 width 属性已经设置了具体的像素值（如 1dp），那么无论该 shape 图形被拉伸到多大，描边宽度始终都是 1dp。点九图片的实现原理与 shape 类似，即拉伸图形时，只对内部进行拉伸，不对边缘做拉伸操作。

为了演示九宫格图片的展示效果，首先我们要制作几张点九图片。Android 的 SDK 自带点九图片的加工工具，路径是 SDK 安装目录下的 sdk\tools\draw9patch.bat，运行该程序就会呈现工具界面，如图 2-25 所示。



图 2-24　普通图片与九宫格图片的拉伸效果对比

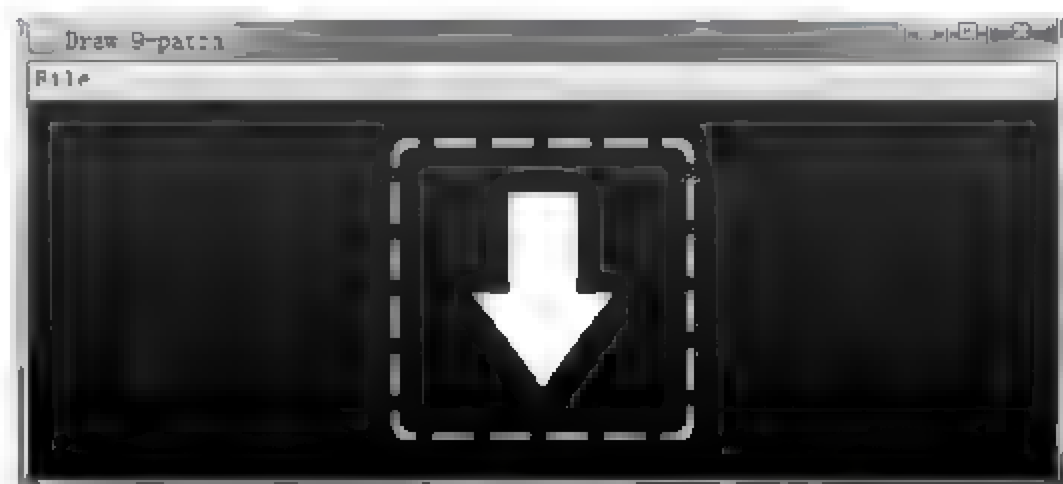


图 2-25　点九图片的制作工具

把需要加工的 PNG 图片拖到该工具界面上，图片就会加载到工具界面，如图 2-26 所示。

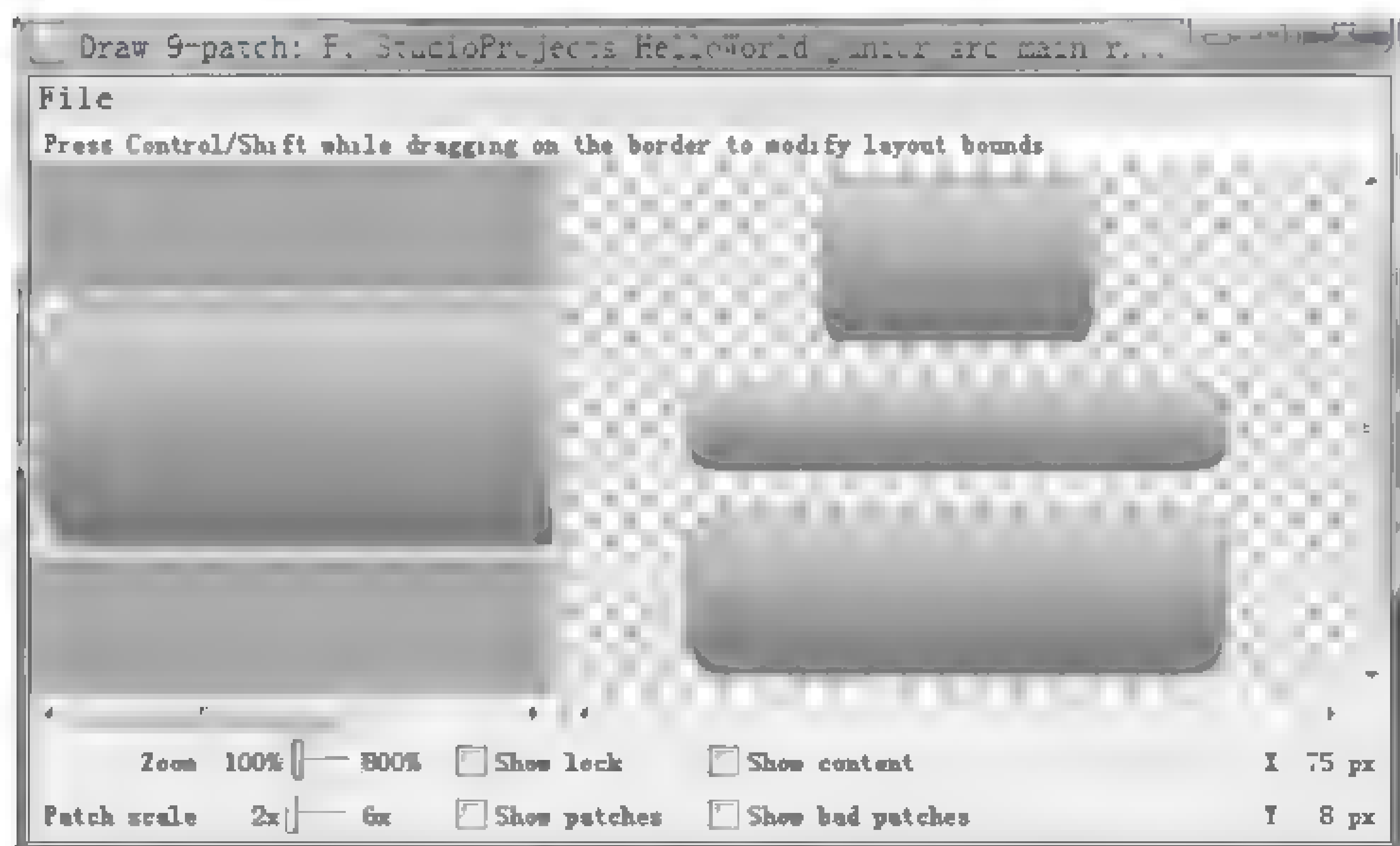


图 2-26　点九图片制作工具的图片加载界面

工具界面的左侧窗口是图片加工区域，右侧窗口是图片预览区域，从上到下依次是纵向拉伸预览、横向拉伸预览、未拉伸预览。在左侧窗口图片四周的马赛克处单击会出现一个黑点，把黑点左右或上下拖动会拖出一段黑线，不同方向上的黑线表示不同的效果。

如图 2-27 所示，界面上边的黑线指的是水平方向的拉伸区域。水平方向拉伸图片时，只有黑线区域内的图像会拉伸，黑线两边的图像保持原状，从而保证左右两边的边框厚度不变。

如图 2-28 所示，界面左边的黑线指的是垂直方向的拉伸区域。垂直方向拉伸图片时，只有黑线区域内的图像会拉伸，黑线两边的图像保持原状，从而保证上下两边的边框厚度不变。

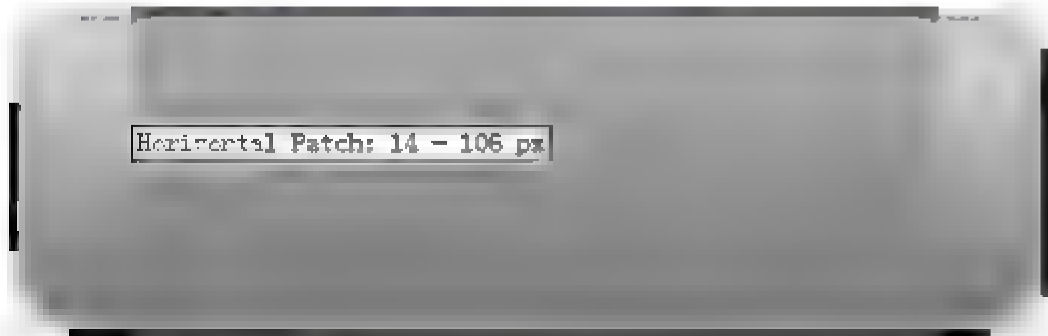


图 2-27　点九图片上边的边缘线



图 2-28　点九图片左边的边缘线

如图 2-29 所示，界面下边的黑线指的是该图片作为控件背景时，控件内部的文字左右边界只能放在黑线区域内。这里 Horizontal Padding 的效果就相当于 android:paddingLeft 与 android:paddingRight。

如图 2-30 所示，界面右边的黑线指的是该图片作为控件背景时，控件内部的文字上下边界只能放在黑线区域内。这里 Vertical Padding 的效果就相当于 android:paddingTop 与 android:paddingBottom。

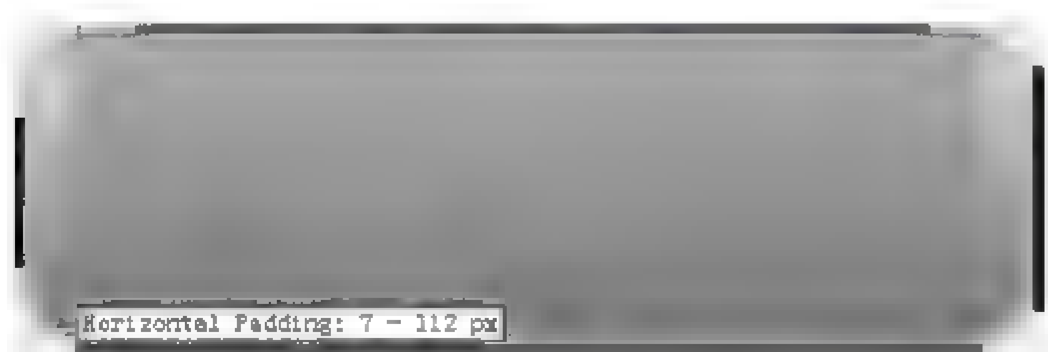


图 2-29　点九图片下边的边缘线



图 2-30　点九图片右边的边缘线

在实际开发中，前两个属性使用的比较多，因为很多场景都要求拉伸图片时要保真。后两个属性一般用得不多，但若不知道，遇到问题还挺麻烦的。笔者以前做开发时看到某个页面的文字总是与顶端有段间隔，可是无论怎么调整 XML 和代码都没法缩小间隔，后来才想起来检查该页面的背景图片，结果用 draw9patch.bat 打开背景图发现该图片是点九图片，原来在水平和垂直方向都设置了 padding，这才解决了一大困惑。

2.5　实战项目：简单计算器

到目前为止，虽然只学了一些 Android 的初级控件，但是也可以学以致用，即便只有这些简单的布局和控件，也能够做出实用的 App。接下来我们设计并实现一个简单计算器。



2.5.1 设计思路

计算器是人们日常生活中最常用的工具之一，无论在电脑上还是手机上，都少不了计算器的身影。以 Windwos 上的计算器为例，界面简洁且十分实用，程序界面如图 2-31 所示。

这个计算器界面主要分为两部分，一部分是上面的文本框，用于显示计算结果；另一部分是下面的几排按钮，用于输入数字与各种运算符。为了减少复杂度，我们可以精简一些功能，只保留数字与加、减、乘、除四则运算，另外补充一个开根号（求平方根）的运算。至于 App 的显示界面，基本与习惯的计算器界面保持一致，经过对操作按钮的适当排列，调整后的设计效果如图 2-32 所示。



图 2-31 Windows 的计算器



图 2-32 简单计算器的设计效果图

这个计算器虽然小巧，但是基本囊括了本章的知识点，先来看看用了哪些控件。

- 线性布局 LinearLayout：计算器界面整体上是从上往下布局的，所以需要垂直方向的 LinearLayout；下面部分每行都有 4 个按钮，又需要水平方向的 LinearLayout。
- 滚动视图 ScrollView：虽然计算器界面不宽也不高，但是以防万一，最好还是加个垂直方向的 ScrollView。
- 文本视图 TextView：很明显上方标题“简单计算器”就是 TextView，下面的计算结果也需要使用 TextView，而且是能够自动从下往上滚动的 TextView，即聊天室效果的文本视图。
- 按钮 Button：绝大多数数字与运算符按钮都采用 Button 控件。
- 图像视图 ImageView：暂时未用到。
- 图像按钮 ImageButton：开根号的运算符“√”虽然能够打出来，但是右上角少了数学课本上的一横，所以该按钮要用一张标准的开根号图片显示，这就用到了 ImageButton。
- 状态列表图形：每个按钮都有按下和弹起两种状态，这里定制了按钮控件的自定义样式，因此用到了状态列表图形。
- 形状图形：运算结果用到的文本视图边框是圆角矩形，所以得给它定义一个 shape 文件，把 shape 定义的圆角矩形作为文本视图的背景。

- 九宫格图片：注意计算器界面左下角的“0”，该按钮是其他按钮的两倍宽，如果使用普通图片当背景，势必造成边缘线被拉宽、拉模糊的问题，故而要采用点九图片避免这种情况。

经过对计算器效果图的详细分析，我们初步了解了所运用的控件技术，接下来就可以对界面进行布局和排列了。

2.5.2 小知识：日志 Log/提示 Toast

在正式编码之前，读者有必要了解一下 Android 中的运行信息调试手段。例如，开发 C 程序时，我们常常用 printf 函数输出程序日志；开发 Java 程序时，我们常常用 System.out.println 函数输出程序日志。同样，App 开发也有相应的函数输出提示信息。提示信息可分为两类，一类是给开发者看的，另一类是给用户看的。

1. Log

给开发者看的提示信息要调用 Log 类的相应方法，日志打印结果可在 Android Studio 界面下方的 logcat 小窗口查看。Log 类各种方法的区别在于日志的等级，具体说明如下。

- Log.e：表示错误信息，比如可能导致程序崩溃的异常。
- Log.w：表示警告信息。
- Log.i：表示一般消息。
- Log.d：表示调试信息，可把程序运行时的变量值打印出来，方便跟踪调试。
- Log.v：表示冗余信息。

2. Toast

给用户看的提示信息要调用 Toast 类的相应方法，提示文字会在屏幕下方以一个小窗口临时展现。对于计算器来说，有好几种情况需要提示用户，如“被除数不能为 0”“开根号的数值不能小于 0”等。

Toast 的简单用法只需一行代码就可以了，示例代码如下：

```
Toast.makeText(MainActivity.this, "提示文字", Toast.LENGTH_SHORT).show();
```

另外，计算器每个按钮的展示风格基本相同，为了减少冗余代码，可将相同的样式定义写在 values 目录下的 styles.xml 文件中，然后在布局文件节点下增加 style="@style/btn_cal"这样的属性定义。下面是 styles.xml 中计算器按钮风格定义的例子：

```
<style name="btn_cal">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:layout_weight">1</item>
    <item name="android:gravity">center</item>
    <item name="android:textColor">@color/black</item>
    <item name="android:textSize">30sp</item>
    <item name="android:background">@drawable/btn_nine_selector</item>
```



```
</style>
```

2.5.3　代码示例



图 2-33　简单计算器的运行效果图

看到这里，估计读者对计算器 App 的布局和代码框架都了然于胸了，接下来介绍一些业务逻辑判断与基本的数学四则运算。只要设计充分并且合理，编码就会很快。计算器 App 运行后的计算效果如图 2-33 所示。

编码过程主要分为 3 个步骤：

步骤 01 先想好代码文件与布局文件的名称，比如代码文件取名 CalculatorActivity.java、布局文件取名 activity_calculator.xml。记得在 AndroidManifest.xml 中注册 acitivity 节点，不然 App 运行时会报 ActivityNotFoundException 异常，具体是在 application 节点下补充一行声明：

```
<activity android:name=".CalculatorActivity" />
```

步骤 02 在 res/layout 目录下创建布局文件 activity_calculator.xml，按照简单计算器的效果图在里面填入各控件的布局结构，并指定相关的属性定义。

步骤 03 在项目的包名目录下创建 CalculatorActivity 类，仿照 MainActivity 代码在 onCreate 内部的 setContentView 方法中填入参数 R.layout.activity_calculator，表示该页面使用 activity_calculator.xml 中定义的界面布局。接着编写具体的控件操作与业务代码。

下面是计算器 App 的主要业务代码片段：

```
public void onClick(View v) {
    int resid = v.getId();
    String inputText;
    if (resid == R.id.ib_sqrt) {
        inputText = "√";
    } else {
        inputText = ((TextView) v).getText().toString();
    }
    Log.d(TAG, "resid="+resid+",inputText="+inputText);
    if (resid == R.id.btn_clear) {
        clear("");
    } else if (resid == R.id.btn_cancel) {
        if (operator.equals("") == true) {
            if (firstNum.length() == 1) {
                firstNum = "0";
            } else if (firstNum.length() > 0) {
                firstNum = firstNum.substring(0, firstNum.length() - 1);
            } else {
                Toast.makeText(this, "没有可取消的数字了", Toast.LENGTH_SHORT).show();
```

```
                    return;
                }
                showText = firstNum;
                tv_result.setText(showText);
            } else {
                if (nextNum.length() == 1) {
                    nextNum = "";
                } else if (nextNum.length() > 0) {
                    nextNum = nextNum.substring(0, nextNum.length() - 1);
                } else {
                    Toast.makeText(this, "没有可取消的数字了", Toast.LENGTH_SHORT).show();
                    return;
                }
                showText = showText.substring(0, showText.length() - 1);
                tv_result.setText(showText);
            }
        } else if (resid == R.id.btn_equal) {
            if (operator.length() == 0 || operator.equals("=") == true) {
                Toast.makeText(this, "请输入运算符", Toast.LENGTH_SHORT).show();
                return;
            } else if (nextNum.length() <= 0) {
                Toast.makeText(this, "请输入数字", Toast.LENGTH_SHORT).show();
                return;
            }
            if (caculate() == true) {
                operator = inputText;
                showText = showText + "=" + result;
                tv_result.setText(showText);
            } else {
                return;
            }
        } else if (resid == R.id.btn_plus || resid == R.id.btn_minus
                || resid == R.id.btn_multiply || resid == R.id.btn_divide ) {
            if (firstNum.length() <= 0) {
                Toast.makeText(this, "请输入数字", Toast.LENGTH_SHORT).show();
                return;
            }
            if (operator.length() == 0 || operator.equals("=") == true || operator.equals("√") == true) {
                operator = inputText;   // 操作符
                showText = showText + operator;
                tv_result.setText(showText);
            } else {
                Toast.makeText(this, "请输入数字", Toast.LENGTH_SHORT).show();
```

```
                return;
            }
        } else if (resid == R.id.ib_sqrt) {
            if (firstNum.length() <= 0) {
                Toast.makeText(this, "请输入数字", Toast.LENGTH_SHORT).show();
                return;
            } else if (Double.parseDouble(firstNum) < 0) {
                Toast.makeText(this, "开根号的数值不能小于 0", Toast.LENGTH_SHORT).show();
                return;
            }
            result = String.valueOf(Math.sqrt(Double.parseDouble(firstNum)));
            firstNum = result;
            nextNum = "";
            operator = inputText;
            showText = showText + " √=" + result;
            tv_result.setText(showText);
            Log.d(TAG, "result="+result+",firstNum="+firstNum+",operator="+operator);
        } else {
            if (operator.equals("＝") == true) {
                operator = "";
                firstNum = "";
                showText = "";
            }
            if (resid == R.id.btn_dot) {
                inputText = ".";
            }
            if (operator.equals("") == true) {
                firstNum = firstNum + inputText;
            } else {
                nextNum = nextNum + inputText;
            }
            showText = showText + inputText;
            tv_result.setText(showText);
        }
    }

    private String operator = "";   // 操作符
    private String firstNum = "";   // 前一个操作数
    private String nextNum = "";   // 后一个操作数
    private String result = "";   // 当前计算结果
    private String showText = "";   // 显示的文本内容
    private boolean caculate() {   // 开始加减乘除四则运算
        if (operator.equals("＋") == true) {
            result = String.valueOf(Arith.add(firstNum, nextNum));
```

```
            } else if (operator.equals("－") == true) {
                result = String.valueOf(Arith.sub(firstNum, nextNum));
            } else if (operator.equals("×") == true) {
                result = String.valueOf(Arith.mul(firstNum, nextNum));
            } else if (operator.equals("÷") == true) {
                if ("0".equals(nextNum)) {
                    Toast.makeText(this, "被除数不能为零", Toast.LENGTH_SHORT).show();
                    return false;
                } else {
                    result = String.valueOf(Arith.div(firstNum, nextNum));
                }
            }
            firstNum = result;
            nextNum = "";
            return true;
        }

        private void clear(String text) {   // 清空并初始化
            showText = text;
            tv_result.setText(showText);
            operator = "";
            firstNum = "";
            nextNum = "";
            result = "";
        }
```

2.6 小　　结

本章主要介绍 App 开发初级控件的相关知识，包括屏幕显示基础（像素、颜色、分辨率）、简单布局的用法（基本视图、线性布局、滚动视图）、简单控件的用法（文本视图、按钮、图像视图、图像按钮）、简单图形的用法（状态列表图形、形状图形、九宫格图片）。最后设计了一个实战项目“简单计算器”，在该项目的 App 编码中运用了前面介绍的大部分简单布局和控件，从而加深了对所学知识的理解；并初步使用 Log 和 Toast，为 App 开发培养良好的编码和调试习惯。

通过本章的学习，读者应该能掌握以下 3 种开发技能：

（1）在布局文件中合理使用本章学到的布局和控件。

（2）在代码中合理调用本章学到的布局和控件的相关方法。

（3）学会制作并使用简单的图形描述文件，包括九宫格图片。





中级控件

本章介绍 App 开发常用的一些中级控件及相关工具，主要包括其他布局用法、特殊按钮的用法、下拉框与基本适配器的用法、编辑框的用法等，另外介绍四大组件之一的 Activity 的基本概念与常见用法。最后结合本章所学的知识演示一个实战项目“登录 App”的设计与实现。

3.1 其他布局

本节介绍 Android 另外两个常用的布局视图，分别是相对布局 RelativeLayout 的属性说明与注意点、框架布局 FrameLayout 的属性说明与注意点。

3.1.1 相对布局 RelativeLayout

RelativeLayout 下级视图的位置是相对位置，得有具体的参照物才能确定最终位置。如果不设定下级视图的参照物，那么下级视图默认显示在 RelativeLayout 内部的左上角。用于确定视图位置的参照物分两种，一种是与该视图自身平级的视图，另一种是该视图的上级视图（RelativeLayout）。与参照物对比，相对位置的属性与类型值见表 3-1。

表 3-1 相对位置的属性与类型的取值说明

XML 中的相对位置属性	RelativeLayout 类的相对位置	相对位置说明
layout_toLeftOf	LEFT_OF	当前视图在指定视图的左边
layout_toRightOf	RIGHT_OF	当前视图在指定视图的右边
layout_above	ABOVE	当前视图在指定视图的上方
layout_below	BELOW	当前视图在指定视图的下方
layout_alignLeft	ALIGN_LEFT	当前视图与指定视图的左侧对齐
layout_alignRight	ALIGN_RIGHT	当前视图与指定视图的右侧对齐
layout_alignTop	ALIGN_TOP	当前视图与指定视图的顶部对齐
layout_alignBottom	ALIGN_BOTTOM	当前视图与指定视图的底部对齐
layout_centerInParent	CENTER_IN_PARENT	当前视图在上级视图中间
layout_centerHorizontal	CENTER_HORIZONTAL	当前视图在上级视图的水平方向居中
layout_centerVertical	CENTER_VERTICAL	当前视图在上级视图的垂直方向居中
layout_alignParentLeft	ALIGN_PARENT_LEFT	当前视图与上级视图的左侧对齐
layout_alignParentRight	ALIGN_PARENT_RIGHT	当前视图与上级视图的右侧对齐
layout_alignParentTop	ALIGN_PARENT_TOP	当前视图与上级视图的顶部对齐
layout_alignParentBottom	ALIGN_PARENT_BOTTOM	当前视图与上级视图的底部对齐

为了更好地理解上述相对属性的含义，接下来使用 RelativeLayout 及其下级视图进行布局，看看实际效果图是怎样的。下面是演示相对布局的 XML 代码：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="500dp" >
    <Button
        android:id="@+id/btn_center"
        style="@style/btn_relative"
```

```
        android:layout_centerInParent="true"
        android:text="我在中间" />
    <Button
        android:id="@+id/btn_center_horizontal"
        style="@style/btn_relative"
        android:layout_centerHorizontal="true"
        android:text="我在水平中间" />
    <Button
        android:id="@+id/btn_center_vertical"
        style="@style/btn_relative"
        android:layout_centerVertical="true"
        android:text="我在垂直中间" />
    <Button
        android:id="@+id/btn_parent_left"
        style="@style/btn_relative"
        android:layout_marginTop="100dp"
        android:layout_alignParentLeft="true"
        android:text="我跟上级左边对齐" />
    <Button
        android:id="@+id/btn_parent_top"
        style="@style/btn_relative"
        android:layout_width="120dp"
        android:layout_alignParentTop="true"
        android:text="我跟上级顶部对齐" />
    <Button
        android:id="@+id/btn_parent_right"
        style="@style/btn_relative"
        android:layout_marginTop="100dp"
        android:layout_alignParentRight="true"
        android:text="我跟上级右边对齐" />
    <Button
        android:id="@+id/btn_parent_bottom"
        style="@style/btn_relative"
        android:layout_width="120dp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="我跟上级底部对齐" />
    <Button
        android:id="@+id/btn_left_bottom"
        style="@style/btn_relative"
        android:layout_toLeftOf="@+id/btn_parent_bottom"
        android:layout_alignTop="@+id/btn_parent_bottom"
        android:text="我在底部左边" />
```

```
    <Button
        android:id="@+id/btn_right_bottom"
        style="@style/btn_relative"
        android:layout_toRightOf="@+id/btn_parent_bottom"
        android:layout_alignBottom="@+id/btn_parent_bottom"
        android:text="我在底部右边" />
    <Button
        android:id="@+id/btn_above_center"
        style="@style/btn_relative"
        android:layout_above="@+id/btn_center"
        android:layout_alignLeft="@+id/btn_center"
        android:text="我在中间上面" />
    <Button
        android:id="@+id/btn_below_center"
        style="@style/btn_relative"
        android:layout_below="@+id/btn_center"
        android:layout_alignRight="@+id/btn_center"
        android:text="我在中间下面" />
</RelativeLayout>
```

上述布局文件的效果如图 3-1 所示，RelativeLayout 的下级视图为各个按钮控件，按钮上的文字说明了所处的相对位置，具体的控件显示方位正如 XML 属性中描述的那样。

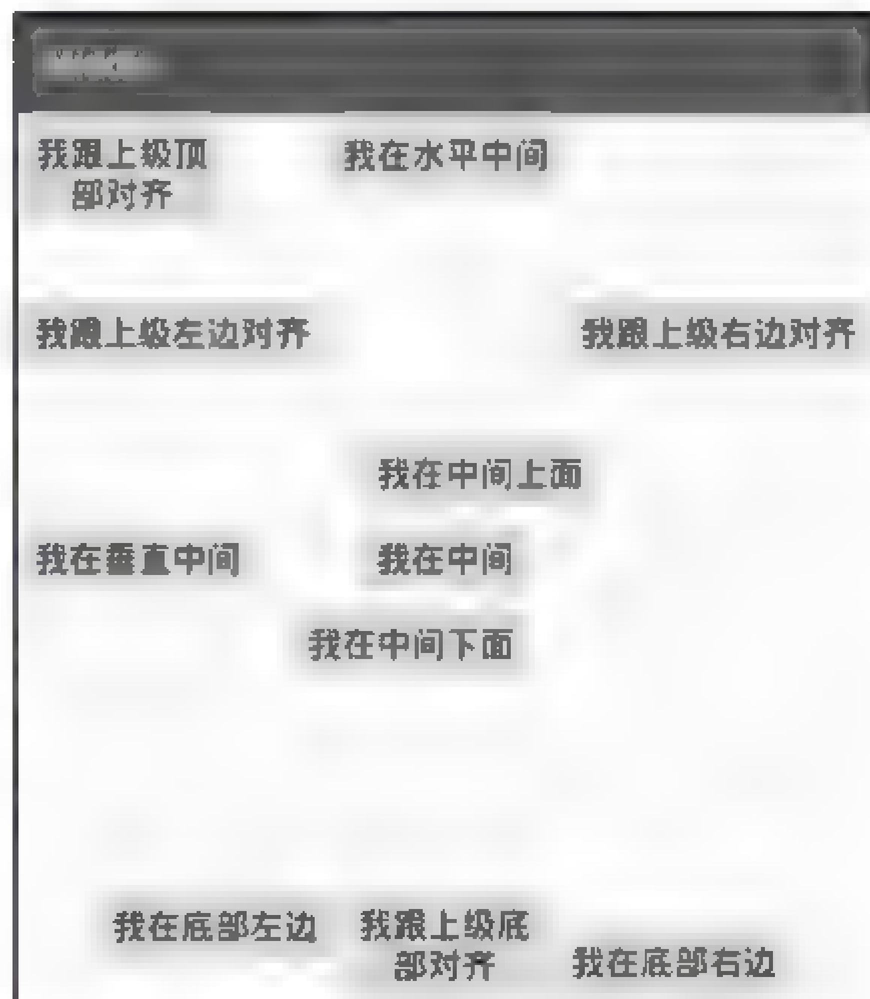


图 3-1　在布局文件中定义的相对布局

一般我们在布局文件中就定义好了视图的相对位置，很少会等到在代码中定义。不过也有特殊情况，如果视图是在代码中动态添加的，那么相对位置也只能在代码中临时定义。代码中定义相对位置用到的是 RelativeLayout.LayoutParams 的 addRule 方法，该方法的第一个参数表示相对位置的类型，具体取值说明见表 3-1；第二个参数表示参照物视图的 ID，即当前视图要参照哪个视图确定自身位置。

下面是在代码中给 RelativeLayout 动态添加子视图并指定子视图相对位置的代码片段：

```
    public void onClick(View v) {
        if (v.getId() == R.id.btn_add_left) {
            addNewView(RelativeLayout.LEFT_OF, RelativeLayout.ALIGN_TOP, v.getId());
        } else if (v.getId() == R.id.btn_add_above) {
            addNewView(RelativeLayout.ABOVE, RelativeLayout.ALIGN_LEFT, v.getId());
        } else if (v.getId() == R.id.btn_add_right) {
            addNewView(RelativeLayout.RIGHT_OF, RelativeLayout.ALIGN_BOTTOM, v.getId());
```

```
            } else if (v.getId() == R.id.btn_add_below) {
                addNewView(RelativeLayout.BELOW, RelativeLayout.ALIGN_RIGHT, v.getId());
            } else if (v.getId() == R.id.btn_add_center) {
                addNewView(RelativeLayout.CENTER_IN_PARENT, -1, rl_content.getId());
            } else if (v.getId() == R.id.btn_add_parent_left) {
                addNewView(RelativeLayout.ALIGN_PARENT_LEFT, RelativeLayout.CENTER_
VERTICAL, rl_content.getId());
            } else if (v.getId() == R.id.btn_add_parent_top) {
                addNewView(RelativeLayout.ALIGN_PARENT_TOP, RelativeLayout.CENTER_
HORIZONTAL, rl_content.getId());
            } else if (v.getId() == R.id.btn_add_parent_right) {
                addNewView(RelativeLayout.ALIGN_PARENT_RIGHT, -1, rl_content.getId());
            } else if (v.getId() == R.id.btn_add_parent_bottom) {
                addNewView(RelativeLayout.ALIGN_PARENT_BOTTOM, -1, rl_content.getId());
            }
        }

        private void addNewView(int firstAlign, int secondAlign, int referId) {
            View v = new View(this);
            v.setBackgroundColor(0xaa66ff66);
            RelativeLayout.LayoutParams rl_params = new RelativeLayout.LayoutParams(100, 100);
            rl_params.addRule(firstAlign, referId);
            if (secondAlign >= 0) {
                rl_params.addRule(secondAlign, referId);
            }
            v.setLayoutParams(rl_params);
            v.setOnLongClickListener(new OnLongClickListener() {
                @Override
                public boolean onLongClick(View vv) {
                    rl_content.removeView(vv);
                    return true;
                }
            });
            rl_content.addView(v);
        }
```

动态添加子控件的效果如图 3-2 所示，在图上给每个方块子视图做了编号，以此区分该方块是由哪个按钮添加的以及添加的相对位置。



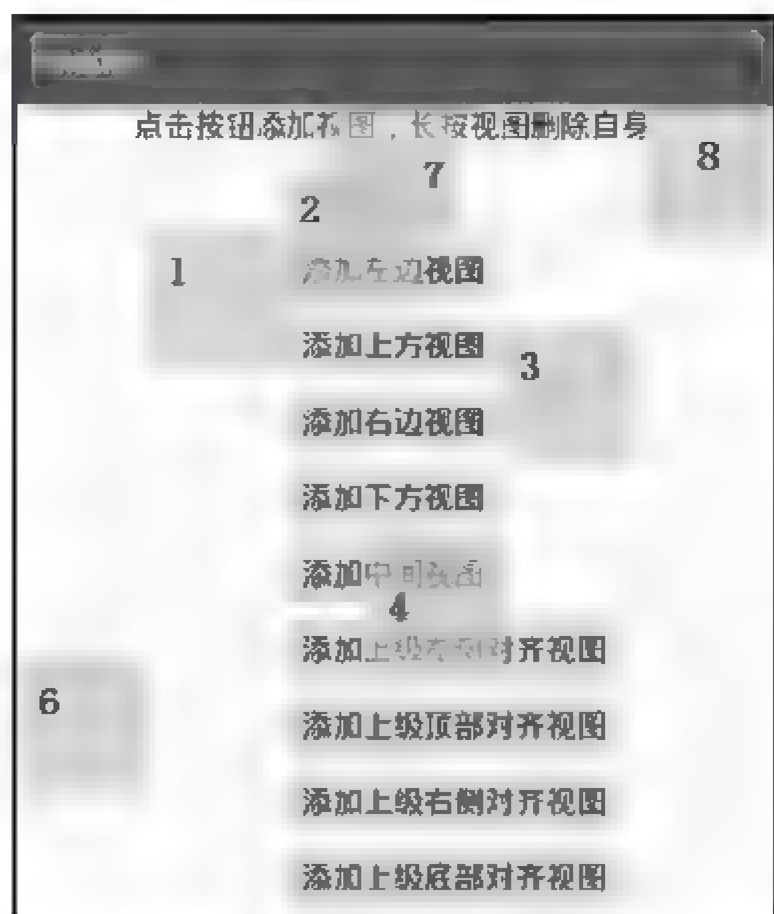


图 3-2　在代码中动态添加下级视图的相对布局

3.1.2　框架布局 FrameLayout

FrameLayout 也是较常用的布局，其下级视图无法指定所处的位置，只能统统从上级 FrameLayout 的左上角开始添加，并且后面添加的子视图会把之前的子视图覆盖掉。框架布局一般用于需要重叠显示的场合，比如绘图、游戏界面等，常见属性说明如下。

- foreground：指定框架布局的前景图像。该图像在框架内部永远处于最顶层，不会被框架内的其他视图覆盖。
- foregroundGravity：指定前景图像的对齐方式。该属性的取值说明同 gravity。

为了更直观地理解 FrameLayout，我们可在代码中为框架布局动态添加子视图，然后观察前后两个子视图的显示效果。

先给框架布局添加一个暗灰色的子视图，如图 3-3 所示。再给框架布局添加一个鲜红色子视图，如图 3-4 所示。此时后面添加的视图会覆盖前面添加的视图。注意，框架视图上方正中间的小图标一直都没被覆盖，是它被指定为前景图像的缘故。

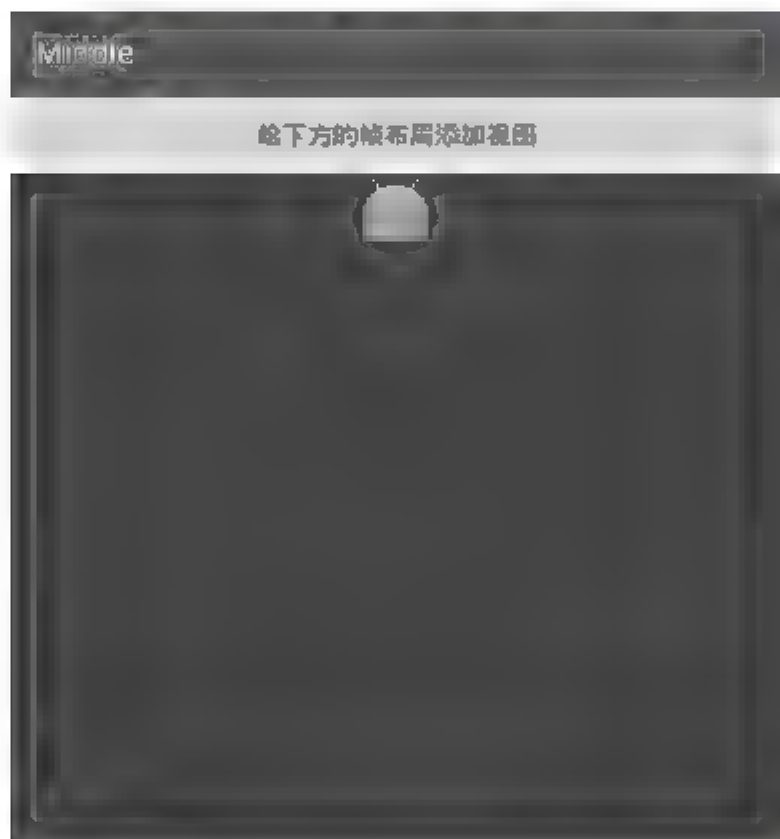


图 3-3　在框架布局中添加第一个子视图



图 3-4　在框架布局中添加第二个子视图

除了线性布局、相对布局、框架布局外，Android 还提供了其他几个布局视图，如绝对布局 AbsoluteLayout、表格布局 TableLayout 等，不过这几个布局在实际开发中用得并不多，读者只需掌握前 3 种布局就可以了。

3.2 特殊按钮

本节介绍几个常用的特殊控制按钮，包括复选框 CheckBox 的监听器用法、开关按钮 Switch 的属性定义、仿 iOS 开关按钮的实现、单选按钮 RadioButton 及其组布局 RadioGroup 的监听器用法，以及如何更换这些控件的按钮图标。

3.2.1 复选框 CheckBox

在学习复选框之前，先了解一下 CompoundButton。在 Android 体系中，CompoundButton 类是抽象的复合按钮，因为是抽象类，所以不能直接使用。实际开发中用的是 CompoundButton 类的几个派生类，主要有复选框 CheckBox、单选按钮 RadioButton 以及开关按钮 Switch，这些派生类都可使用 CompoundButton 的属性和方法。

CompoundButton 在布局文件中主要使用下面两个属性。

- checked：指定按钮的勾选状态，true 表示勾选，false 表示未勾选。默认未勾选。
- button：指定左侧勾选图标的图形。如果不指定就使用系统的默认图标。

CompoundButton 在代码中可使用下列 4 种方法进行设置。

- setChecked：设置按钮的勾选状态。
- setButtonDrawable：设置左侧勾选图标的图形。
- setOnCheckedChangeListener：设置勾选状态变化的监听器。
- isChecked：判断按钮是否勾选。

复选框 CheckBox 是 CompoundButton 一个最简单的实现，点击复选框勾选，再次点击取消勾选。CheckBox 通过 setOnCheckedChangeListener 方法设置勾选监听器，对应的监听器要实现接口 CompoundButton.OnCheckedChangeListener。下面是复选框自定义勾选监听器的代码：

```
    private class CheckListener implements CompoundButton.OnCheckedChangeListener{
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            String desc = String.format("您勾选了控件%d，状态为%b", buttonView.getId(),
isChecked);
            Toast.makeText(MainActivity.this, desc, Toast.LENGTH_LONG).show();
        }
    }
```

要更换复选框左侧的勾选图像，可将 button 属性修改为自定义的勾选图形。下面是一个勾选图形状态定义的例子，如果是勾选状态，就显示图形 check_choose；如果取消勾选，就显



示图形 check_unchoose。

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/check_choose"/>
    <item android:drawable="@drawable/check_unchoose"/>
</selector>
```

3.2.2 开关按钮 Switch

Switch 是开关按钮，Android 从 4.1.2 版本开始支持该控件。其实 Switch 是一个高级版本的 CheckBox，在选中与取消选中时可展现的界面元素比 CheckBox 丰富。Switch 新添加的属性和设置方法见表 3-2。

表 3-2 Switch 控件的属性和设置方法说明

XML 中的属性	Switch 类的设置方法	说明
textOn	setTextOn	设置右侧开启时的文本
textOff	setTextOff	设置左侧关闭时的文本
switchPadding	setSwitchPadding	设置左右两个开关按钮之间的距离
thumbTextPadding	setThumbTextPadding	设置文本左右两边的距离。如果设置了该属性，switchPadding 属性就会失效
thumb	setThumbDrawable setThumbResource	设置开关轨道的背景
track	setTrackDrawable setTrackResource	设置开关标识的图标

Switch 是升级版的 CheckBox，实际开发中用得不多。原因之一是大家觉得 Switch 的默认界面很丑，如图 3-5 和图 3-6 所示，方方正正的图标有点土又有点呆板；原因之二是 iPhone 作为高大上手机的代表，大家都觉得 iOS 的 UI 很漂亮，于是无论是用户还是客户，都希望 App 做得与 iOS 控件相像，iOS 的开关按钮 UISwitch 就成了大家仿照的对象。



图 3-5 Switch 控件的“关”状态



图 3-6 Switch 控件的“开”状态

现在我们要让 Android 实现类似 iOS 的开关按钮，主要思路是借助状态列表图形 StateListDrawable，首先定义一个状态列表，XML 的代码如下：

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_checked="true" android:drawable="@drawable/switch_on"/>
    <item android:drawable="@drawable/switch_off"/>
</selector>
```



然后把 CheckBox 控件的 background 属性设置为该状态图形，当然 button 属性要先设置为@null。为什么这里修改 background 属性，而不直接修改 button 属性呢？因为 button 属性是有限制的，无论多大的图片，都只显示一个小小的图标，可是小小的图标怎么能体现用户高大上的身份呢？所以这里必须使用 background，要它有多大就能有多大，这才够炫、够档次。

最后看看这个仿 iOS 开关按钮的效果，如图 3-7 和图 3-8 所示。这下开关按钮脱胎换骨，又圆又鲜艳，看起来好看很多。



图 3-7　仿 iOS 按钮的“关”状态



图 3-8　仿 iOS 按钮的“开”状态

3.2.3　单选按钮 RadioButton

单选按钮要在一组按钮中选择其中一项，并且不能多选，这要求有个容器确定这组按钮的范围，这个容器便是 RadioGroup。RadioGroup 实质上是个布局，同一组 RadioButton 都要放在同一个 RadioGroup 节点下。RadioGroup 有 orientation 属性可指定下级控件的排列方向，该属性为 horizontal 时，单选按钮在水平方向排列；该属性为 vertical 时，单选按钮在垂直方向排列。RadioGroup 下面除了 RadioButton，还可以挂载其他子控件（如 TextView、ImageView 等）。这样看来，RadioGroup 就是一个特殊的线性布局，只不过多了管理单选按钮的功能。

下面是 RadioGroup 常用的 3 个方法。

- check：选中指定资源编号的单选按钮。
- getCheckedRadioButtonId：获取选中状态单选按钮的资源编号。
- setOnCheckedChangeListener：设置单选按钮勾选变化的监听器。

RadioButton 默认未选中，点击后显示选中，但是再次点击不会取消选中。只有点击同组的其他单选按钮时，原来选中的单选按钮才会取消选中。另外，单选按钮的选中事件一般不由 RadioButton 处理，而是由 RadioGroup 响应。选中事件在实现时，首先要写一个选中监听器实现接口 RadioGroup.OnCheckedChangeListener，然后调用 RadioGroup 对象的 setOnCheckedChangeListener 方法注册该监听器。

下面是用 RadioGroup 实现选中监听器的代码：

```
    class RadioListener implements RadioGroup.OnCheckedChangeListener{
        @Override
        public void onCheckedChanged(RadioGroup group, int checkedId) {
            Toast.makeText(MainActivity.this, "您选中了控件"+checkedId,
Toast.LENGTH_LONG).show();
        }
    }
```

RadioButton 经常会更换按钮图标，如果通过 button 属性变更图标，那么图标与文字就会

挨得很近，如图 3-9 所示的第一个单选按钮。为了拉开图标与文字之间的距离，得换成 drawableLeft 属性展示新图标（不要忘了把 button 改为@null），此时再设置 drawablePadding 即可指定间隔距离。修改后的单选按钮效果如图 3-10 所示，可以看到图标与文字之间的距离明显增大了。

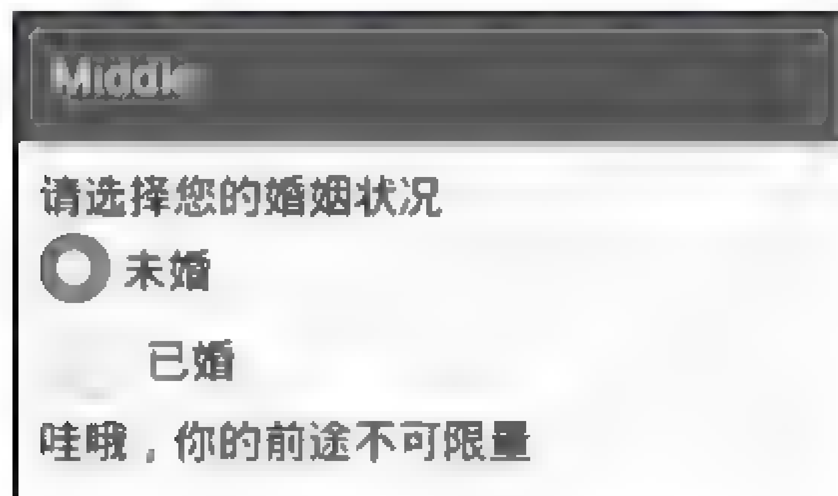


图 3-9　图标设置在 button 属性上

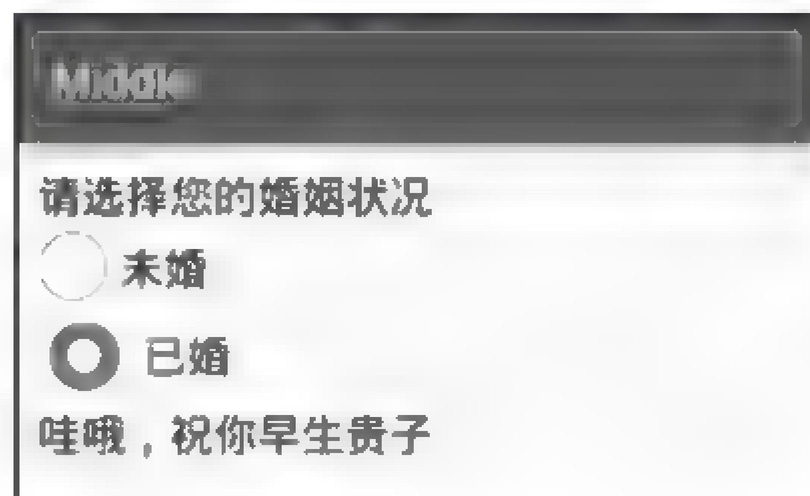


图 3-10　图标设置在 drawableLeft 属性上

前面给不同的按钮自定义按钮图标先后用了 3 个属性，即自定义 CheckBox 图标时的 button 属性、仿 iOS 开关按钮时的 background 属性以及自定义 RadioButton 时的 drawableLeft 属性。下面总结一下这 3 个图标设置方式分别适用的场合。

- button：主要用于图标大小要求不高，间隔要求也不高的场合。
- background：主要用于能够以较大空间显示图标的场合。
- drawableLeft：主要用于对图标与文字之间的间隔有要求的场合。

3.3　适配视图基础

本节介绍适配器的基本概念，结合对下拉框 Spinner 的使用说明分别阐述数组适配器 ArrayAdapter、简单适配器 SimpleAdapter 的具体用法与展示效果。

3.3.1　下拉框 Spinner

Spinner 是下拉框，用于从一串列表中选择某项，功能类似于单选按钮的组合。下拉列表的展示方式有两种，一种是在当前下拉框的正下方展示列表，此时把 spinnerMode 属性设置为 dropdown；另一种是在页面中部以对话框形式展示列表，此时把 spinnerMode 属性设置为 dialog。另外，Spinner 还可以在代码中调用下列 4 个方法。

- setPrompt：设置标题文字。
- setAdapter：设置下拉列表的适配器。适配器可选择 ArrayAdapter 或 SimpleAdapter。
- setSelection：设置当前选中哪项。注意该方法要在 setAdapter 方法后调用。
- setOnItemSelectedListener：设置下拉列表的选中监听器，该监听器要实现接口 OnItemSelectedListener。

下面是一个自定义选中监听器的例子：



```
        private String[] starArray = {"水星", "金星", "地球", "火星", "木星", " 土星"};
        private class MySelectedListener implements OnItemSelectedListener {
                public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
                        Toast.makeText(SpinnerDialogActivity.this, "您选择的是"+starArray[arg2],
Toast.LENGTH_LONG).show();
                }

                public void onNothingSelected(AdapterView<?> arg0) {
                }
        }
```

下面是使用 Spinner 控件的代码片段：

```
                ArrayAdapter<String> starAdapter = new ArrayAdapter<String>(this,
                                R.layout.item_select, starArray);
                starAdapter.setDropDownViewResource(R.layout.item_dropdown);
                Spinner sp = (Spinner) findViewById(R.id.sp_dialog);
                sp.setPrompt("请选择行星");
                sp.setAdapter(starAdapter);
                sp.setSelection(0);
                sp.setOnItemSelectedListener(new MySelectedListener());
```

接下来看对话框模式的下拉效果，如图 3-11 所示。页面中部弹出六大行星的下拉列表；点击具体行星项后自动收起下拉列表，并且下拉框中的文字变更为刚选中的行星名称。

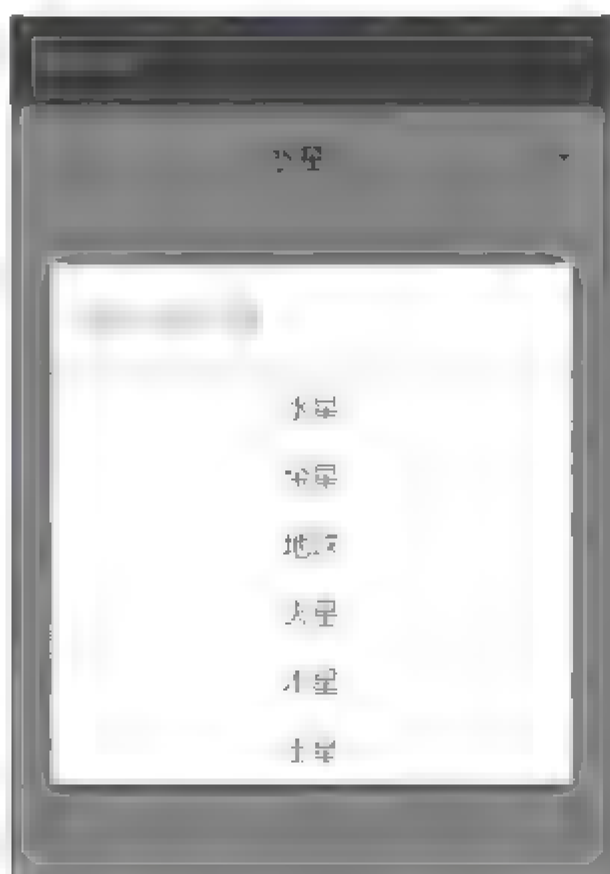


图 3-11　dialog 模式的下拉列表

3.3.2　数组适配器 ArrayAdapter

前面在演示 Spinner 时用到了 setAdapter 方法设置适配器。这个适配器好比一组数据的加工流水线，你丢给它一大把糖果，适配器把糖果排列好顺序，然后拿来制作好的包装盒，把糖果往里面一塞，出来的便是一个个精美的糖果盒。这个流水线可以做得很复杂，也可以做得简单一些，最简单的流水线就是之前演示 Spinner 用到的 ArrayAdapter。

ArrayAdapter 主要用于每行列表只展示文本的情况，有两道工序，第一道工序是构造函数，除了提供一堆原始数据外（六大行星的名称列表），还可以指定下拉框当前文本的包装盒，即下面这行代码里的 R.layout.item_select，这个布局文件内只有一个 TextView，定义了当前选中文本的大小、颜色、对齐方式等属性。

```
ArrayAdapter<String> starAdapter = new ArrayAdapter<String>(this,
        R.layout.item_select, starArray);
```

第二道工序是定义下拉列表的包装盒，即下面代码里的 R.layout.item_dropdown，定义了对话框列表中每行文本的显示属性。

```
starAdapter.setDropDownViewResource(R.layout.item_dropdown);
```

经过这两道工序，ArrayAdapter 就明确了原料糖果的分拣过程与包装方式，接下来只待 Spinner 调用 setAdapter 方法发出开动机器指令，适配器便会把一个一个糖果盒输出到屏幕界面。

3.3.3 简单适配器 SimpleAdapter

ArrayAdapter 只能显示文本列表，显然不够美观，有时我们还想给列表加上图标，比如六大行星是否分别显示星球的小图。这时 SimpleAdapter 就派上用场了，它允许在列表项中展示多个控件，包括文本与图片。

SimpleAdapter 的实现略微复杂，除了第二道工序与 ArrayAdapter 一样外，第一道工序需要更多信息。例如，原料不但有糖果，还有贺卡，这样就得把一大袋糖果和一大袋贺卡送进流水线，适配器每次拿一颗糖果和一张贺卡，把糖果与贺卡按规定塞进包装盒。对于 SimpleAdapter 的构造函数来说，第二个参数 Map 容器放的是原料糖果与贺卡，第 3 个参数放的是包装盒，第 4 个参数放的是糖果袋与贺卡袋的名称，第 5 个参数放的是包装盒里塞糖果的位置与塞贺卡的位置。

下面是使用 SimpleAdapter 的示例代码：

```
int[] iconArray = {R.drawable.shuixing, R.drawable.jinxing, R.drawable.diqiu,
      R.drawable.huoxing, R.drawable.muxing, R.drawable.tuxing };
List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
for (int i = 0; i < iconArray.length; i++) {
      Map<String, Object> item = new HashMap<String, Object>();
      item.put("icon", iconArray[i]);
      item.put("name", starArray[i]);
      list.add(item);
}
SimpleAdapter starAdapter = new SimpleAdapter(this, list, R.layout.item_select,
        new String[] { "icon", "name" }, new int[] { R.id.iv_icon, R.id.tv_name });
starAdapter.setDropDownViewResource(R.layout.item_simple);
```

下面是每个列表项的布局文件代码（包装盒）：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <ImageView
            android:id="@+id/iv_icon"
            android:layout_width="0dp"
            android:layout_height="50dp"
            android:layout_weight="1"
            android:gravity="center" />

        <TextView
            android:id="@+id/tv_name"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="3"
            android:gravity="center"
            android:textSize="17sp"
            android:textColor="#ff0000" />
    </LinearLayout>
```

敲了这么多代码，下面看一下加了图标的下拉列表的效果图，如图 3-12 所示。此时下拉列表左边显示行星的图片，右边显示行星的名称。

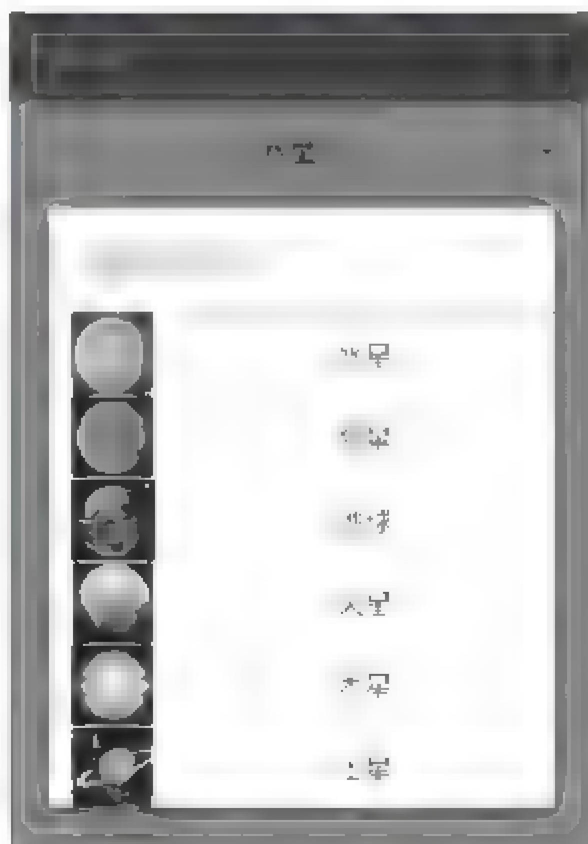


图 3-12　带图标的下拉列表

3.4 编 辑 框

本节介绍 Android 的两种编辑框，分别是文本编辑框 EditText 与自动完成编辑框 AutoCompleteTextView。在介绍 EditText 控件时，除了基本属性和方法，还另外阐述了常见的 4 种编辑处理：更换光标、更换边框、自动隐藏输入法和输入回车符自动换行。

3.4.1 文本编辑框 EditText

EditText 是文本编辑框，用户可在此输入文本等信息。EditText 的常用属性说明如下。

- inputType：指定输入的文本类型，代码中对应的方法是 setInputType。输入类型的取值说明见表 3-3，若同时使用多种文本类型，则可使用竖线"|"把多种文本类型拼接起来。
- maxLength：指定文本允许输入的最大长度。该属性无法通过代码设置。
- hint：指定提示文本的内容，代码中对应的方法是 setHint。
- textColorHint：指定提示文本的颜色，代码中对应的方法是 setHintTextColor。

表 3-3 输入类型的取值说明

输入类型	说明
text	文本
textPassword	文本密码。显示时用星号"*"代替
number	整型数
numberSigned	带符号的数字。允许在开头带负号"－"
numberDecimal	带小数点的数字
numberPassword	数字密码。显示时用星号"*"代替
datetime	时间日期格式。除了数字外，还允许输入横线、斜杆、空格、冒号
date	日期格式。除了数字外，还允许输入横线"-"和斜杆"/"
time	时间格式。除了数字外，还允许输入冒号":"

编辑框除了上述文本与提示文本的基本操作外，实际开发中还常常关注 4 个方面：更换编辑框的光标、更换编辑框的边框、自动隐藏输入法、输入回车符自动跳转。

1. 更换编辑框的光标

EditText 与光标处理有关的属性主要有两个，分别是：

- cursorVisible，指定光标是否可见。代码中对应的方法是 setCursorVisible。
- textCursorDrawable，指定光标的图像。该属性无法通过代码设置。

如果要隐藏光标，就要把 cursorVisible 设置为 false。如果要变更光标的样式，就要修改 textCursorDrawable 设置新图像。如图 3-13 所示，光标被换成自定义的红色竖线光标。

2. 更换编辑框的边框

EditText 的边框通过 background 属性控制，如果要隐藏边框，就要把 background 设置为 @null；如果要修改边框的样式，就要将 background 设置为其他边框图形。

下面是一个边框定义 XML 的例子，一旦编辑框获得焦点（例如用户点击了该编辑框），边框就会显示图形 shape_edit_focus；否则默认显示 shape_edit_normal。

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_focused="true" android:drawable="@drawable/shape_edit_focus"/>
```



```
    <item android:drawable="@drawable/shape_edit_normal"/>
</selector>
```

上述自定义边框的效果如图 3-14 所示，未点击时显示灰色的圆角边框，点击后显示蓝色的圆角边框。

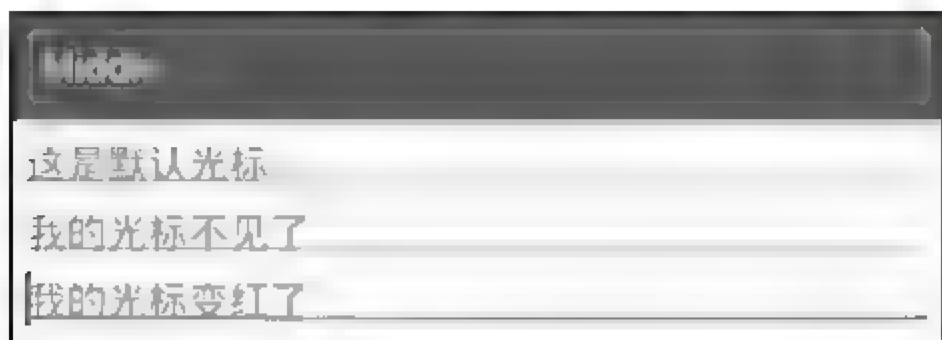


图 3-13　给 EditText 更换图标样式



图 3-14　给 EditText 更换边框样式

3. 自动隐藏输入法

如果页面上有 EditText 控件，开发者又没做其他处理，那么用户打开该页面时往往会自动弹出输入法。这是因为编辑框会默认获得焦点，即默认模拟用户的点击操作，于是输入法的软键盘就弹出了。要想避免这种情况，就得阻止编辑框默认获得焦点。比较常见的做法是给该页面的根节点设置 focusable 和 focusableInTouchMode 属性，通过将这两个属性设置为 true 可强制让根节点获得焦点，从而避免输入法自动弹出的尴尬。

由于软键盘通常会遮盖“登录”“确认”“下一步”等按钮，造成用户输入完毕得再点一次返回键才能关闭软键盘。大家都希望省事点，比如手机号输入满 11 位软键盘自动关闭，这样就会极大改善用户体验。一个好用的 App 就是在这一点一滴中体现出来的。

想让编辑框文本达到指定长度时自动关闭输入法，开发者需要获得两个参数，第一个是该编辑框允许输入的最大长度，第二个是当前已经输入的文本长度。当已输入的文本长度等于最大长度时，即可触发关闭软键盘。自动隐藏输入法可分解为 3 个功能点，分别是获取编辑框的最大长度、监控当前已输入的文本长度和关闭软键盘。

（1）获取编辑框的最大长度

前面我们了解到 maxLength 属性可设置最大长度，但是 EditText 并没有提供获取最大长度的方法，不过我们可以通过反射方式曲线获得最大长度，具体代码如下：

```
public static int getMaxLength(EditText et) {
    int length = 0;
    try {
        InputFilter[] inputFilters = et.getFilters();
        for (InputFilter filter : inputFilters) {
            Class<?> c = filter.getClass();
            if (c.getName().equals("android.text.InputFilter$LengthFilter")) {
                Field[] f = c.getDeclaredFields();
                for (Field field : f) {
                    if (field.getName().equals("mMax")) {
                        field.setAccessible(true);
```

```
                                length = (Integer) field.get(filter);
                            }
                        }
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return length;
        }
```

（2）监控当前已输入的文本长度

这个监控操作用到一个文本监听器接口 TextWatcher，该接口提供了 3 个监控方法，具体说明如下。

- beforeTextChanged：在文本改变之前触发。
- onTextChanged：在文本改变过程中触发。
- afterTextChanged：在文本改变之后触发。

这里用到的是 afterTextChanged 方法，开发者需要自己写个监听器实现 TextWatcher 接口，另外再给 EditText 对象调用 addTextChangedListener 方法注册该监听器。下面是一个具体实现该监听器的例子：

```
    private class HideTextWatcher implements TextWatcher {
        private EditText mView;
        private int mMaxLength;
        private CharSequence mStr;

        public HideTextWatcher(EditText v) {
            super();
            mView = v;
            mMaxLength = ViewUtil.getMaxLength(v);
        }

        @Override
        public void beforeTextChanged(CharSequence s, int start, int count, int after) {
        }

        @Override
        public void onTextChanged(CharSequence s, int start, int before, int count) {
            mStr = s;
        }
```



```
        @Override
        public void afterTextChanged(Editable s) {
            if (mStr == null || mStr.length() == 0)
                return;
            if (mStr.length() == 11 && mMaxLength == 11) {
                ViewUtil.hideAllInputMethod(EditHideActivity.this);
            } else if (mStr.length() == 6 && mMaxLength == 6) {
                ViewUtil.hideOneInputMethod(EditHideActivity.this, mView);
            }
        }
    }
```

（3）关闭软键盘

输入法通过系统服务 INPUT_METHOD_SERVICE 管理，所以隐藏输入法也要通过该服务实现。下面是关闭软键盘的两种方式及其代码：

① 调用 toggleSoftInput 方法：

```
    public static void hideAllInputMethod(Activity act) {
        InputMethodManager imm = (InputMethodManager)
act.getSystemService(Context.INPUT_METHOD_SERVICE);
        if (imm.isActive() == true) {   //软键盘如果已经打开就要关闭
            imm.toggleSoftInput(0, InputMethodManager.HIDE_NOT_ALWAYS);
        }
    }
```

② 调用 hideSoftInputFromWindow 方法：

```
    public static void hideOneInputMethod(Activity act, View v) {
        InputMethodManager imm = (InputMethodManager)
act.getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(v.getWindowToken(), 0);
    }
```

完成隐藏输入法的编码后，可在页面上观察效果，如图 3-15 所示。此时手机号码输入了 10 位，还没达到 11 位的最大长度，故而输入法依然显示。手机号再输入一位数字，总长度 11 位达到最大长度的限制，于是输入法自动隐藏，如图 3-16 所示。

4. 输入回车符自动跳转

在录入用户信息时（比如输入姓名、密码等），往 EditText 控件输入回车键，常常不是换行而是让光标直接跳到下一个编辑框。该功能用到了文本监听器接口 TextWatcher，主要监听用户是否输入回车符，如果监控到已输入回车符，就自动将焦点移到下一个控件，从而实现回车符自动跳转的要求。



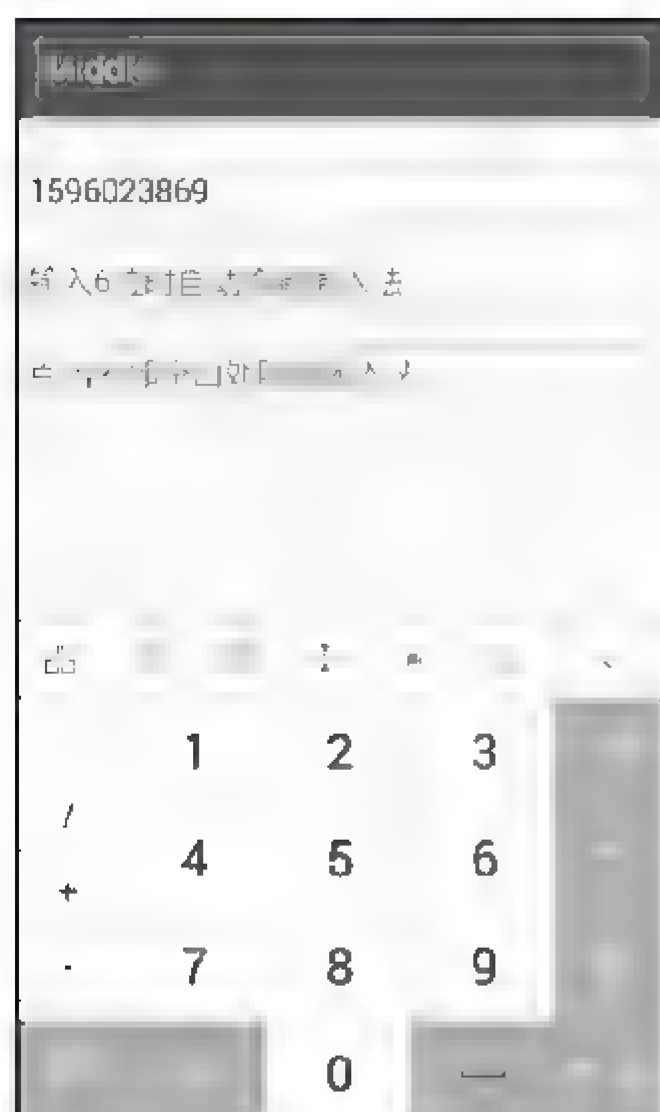


图 3-15　输入 10 位手机号码

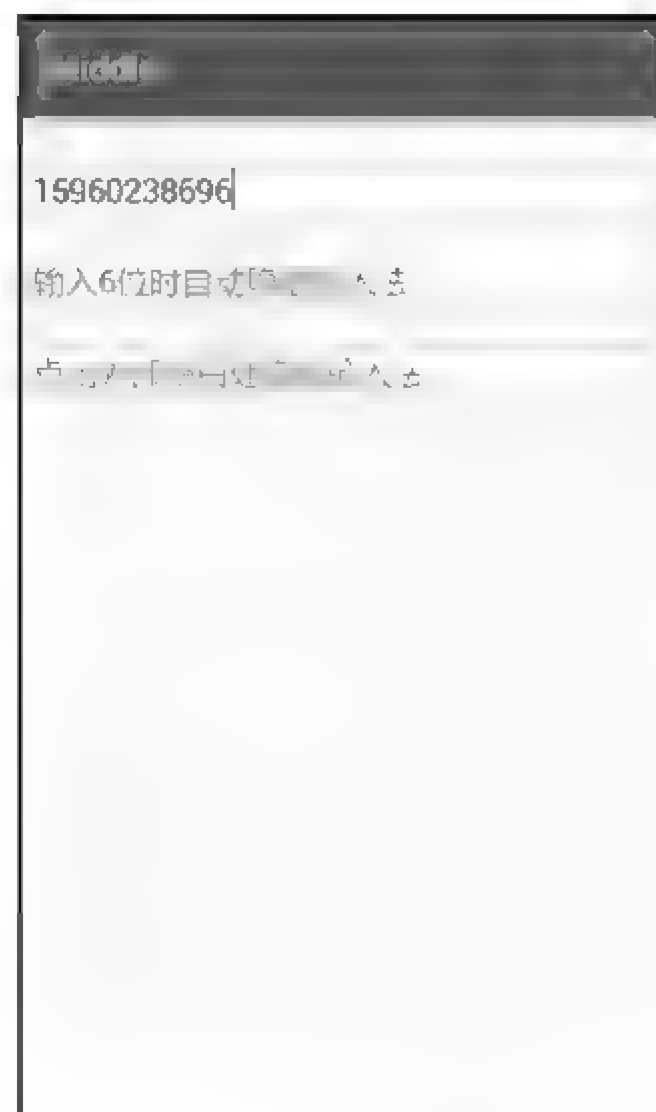


图 3-16　输入 11 位手机号码

下面是回车符监听器的代码，注意注释部分的文字说明：

```
private class JumpTextWatcher implements TextWatcher {
    private EditText mThisView = null;
    private View mNextView = null;

    public JumpTextWatcher(EditText vThis, View vNext) {
        super();
        mThisView = vThis;
        if (vNext != null) {
            mNextView = vNext;
        }
    }

    @Override
    public void beforeTextChanged(CharSequence s, int start, int count, int after) {
    }

    @Override
    public void onTextChanged(CharSequence s, int start, int before, int count) {
    }

    @Override
    public void afterTextChanged(Editable s) {
        String str = s.toString();
        if (str.indexOf("\r") >= 0 || str.indexOf("\n") >= 0) {    //发现输入回车符或换行符
```

```
                    mThisView.setText(str.replace("\r", "").replace("\n", ""));  //去掉回车符和换行符
                    if (mNextView != null) {
                        mNextView.requestFocus();  //让下一个视图获得焦点，即将光标移到下个视图
                        if (mNextView instanceof EditText) {
                            EditText et = (EditText)mNextView;
                            //让光标自动移到编辑框内部的文本末尾
                            //方式一：直接调用 EditText 的 setSelection 方法
                            et.setSelection(et.getText().length());
                            //方式二：调用 Selection 类的 setSelection 方法
                            //Editable edit = et.getText();
                            //Selection.setSelection(edit, edit.length());
                        }
                    }
                }
            }
        }
```

下面演示一下输入回车符自动跳转的效果图，文本输入完毕后还没输入回车符，此时焦点仍然停留在编辑框，如图 3-17 所示。输入回车符，此时焦点离开编辑框，并自动移动到“登录”按钮（编辑框的光标消失，按钮背景变深），如图 3-18 所示。



图 3-17　未按回车符



图 3-18　已按回车符

3.4.2　自动完成编辑框 AutoCompleteTextView

自动完成编辑框一般用于搜索文本框，如在电商 App 的搜索框输入商品文字时，下方会自动弹出提示词列表，方便用户快速选择具体商品。AutoCompleteTextView 的实现原理是：EditText 结合监听器 TextWatcher 与下拉列表 Spinner，一旦监控到 EditText 的文本发生变化，就自动弹出适配好的文字下拉列表，选中具体的下拉项向 EditText 填入相应文字。

AutoCompleteTextView 新增的几个属性都与下拉列表有关，详细说明见表 3-4。

表 3-4　自动完成编辑框的属性和设置方法说明

XML 中的属性	AutoCompleteTextView 类的设置方法	说明
completionHint	setCompletionHint	设置下拉列表底部的提示文字
completionThreshold	setThreshold	设置至少输入多少个字符才会显示提示
dropDownHorizontalOffset	setDropDownHorizontalOffset	设置下拉列表与文本框之间的水平偏移
dropDownVerticalOffset	setDropDownVerticalOffset	设置下拉列表与文本框之间的垂直偏移

（续表）

XML 中的属性	AutoCompleteTextView 类的设置方法	说明
dropDownHeight	setDropDownHeight	设置下拉列表的高度
dropDownWidth	setDropDownWidth	设置下拉列表的宽度
无	setAdapter	设置下拉列表的数据适配器

下面是使用 AutoCompleteTextView 的代码：

```
String[] hintArray = {"第一", "第一次", "第一次写代码", "第一次领工资", "第二", "第二个"};
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        this, R.layout.item_dropdown, hintArray);
AutoCompleteTextView ac_text= (AutoCompleteTextView) findViewById(R.id.ac_text);
ac_text.setAdapter(adapter);
```

自动完成编辑框的具体效果如图 3-19 所示，下拉列表的内容会自动与编辑框的文本进行匹配。



图 3-19　自动完成编辑框的自动匹配下拉列表

3.5　Activity 基础

本节介绍 Android 四大组件之一 Activity 的基本概念和常见用法。首先说明 Activity 的生命周期，接着说明 Intent 的组成部分与工作原理，然后阐述如何使用 Intent 完成活动页面之间的消息传递，包括如何传递请求参数、如何返回应答参数等。

3.5.1　Activity 的生命周期

看到这里，相信读者对 Activity 已经不陌生了。首先，一个 Activity 代表一个页面。其次，Activity 的 onCreate 方法是页面的入口函数。更细心的读者也许已经知道调用 startActivity 方法可以跳转到下一个页面。之所以到这时才介绍 Activity，是因为 Activity 的逻辑复杂、概念繁多，必须在有一定基础后讲解才合适，不然一开始就讲解高深的专业术语，读者恐怕很难理解。



首先介绍 Activity 的生命周期，如同花开花落一般，Activity 也有从含苞待放到盛开再到凋零的生命过程。下面是 Activity 与生命周期有关的方法说明。

- onCreate：创建页面。把页面上的各个元素加载到内存中。
- onStart：开始页面。把页面显示在屏幕上。
- onResume：恢复页面。让页面在屏幕上活动起来，例如开启动画、开始任务等。
- onPause：暂停页面。让页面在屏幕上的动作停下来。
- onStop：停止页面。把页面从屏幕上撤下来。
- onDestroy：销毁页面。把页面从内存中清除掉。
- onRestart：重启页面。重新加载内存中的页面数据。

下面针对几个常见的业务场景探究一下 Activity 的生命周期，主要有 3 个场景：页面之间的跳转、竖屏与横屏的切换、按 HOME 键与返回 App。用于场景测试的代码如下，主要在每个生命周期函数中增加打印屏幕日志和后台日志。

```
private void refreshLife(String desc) {
    Log.d(TAG, desc);
    mStr = String.format("%s%s %s %s\n", mStr, DateUtil.getNowTimeDetail(), TAG, desc);
    tv_life.setText(mStr);
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_act_***);
    tv_life = (TextView) findViewById(R.id.tv_life);
    refreshLife("onCreate");
}

@Override
protected void onStart() {
    refreshLife("onStart");
    super.onStart();
}

@Override
protected void onStop() {
    refreshLife("onStop");
    super.onStop();
}

@Override
protected void onResume() {
```

```
        refreshLife("onResume");
        super.onResume();
    }

    @Override
    protected void onPause() {
        refreshLife("onPause");
        super.onPause();
    }

    @Override
    protected void onRestart() {
        refreshLife("onRestart");
        super.onRestart();
    }

    @Override
    protected void onDestroy() {
        refreshLife("onDestroy");
        super.onDestroy();
    }
```

1. 页面之间的跳转

首先进入测试页面 ActJumpActivity，接着从该页面跳转到 ActNextActivity，然后从 ActNextActivity 返回 ActJumpActivity。界面上的日志截图如图 3-20 所示。其中，区域 1 表示进入页面 ActJumpActivity 时的生命周期过程，区域 2 表示跳转到 ActNextActivity 时的生命周期过程，区域 3 表示返回 ActJumpActivity 时的生命周期过程。

图 3-20　活动页面跳转时的界面日志截图

从日志截图可以看到，下一个页面的创建伴随上一个页面的停止，不过显示的日志信息不够完整。下面我们跟踪一下 logcat 里的日志，看看这中间到底发生了什么。

首先打开页面 ActJumpActivity，调用方法的顺序为：本页面 onCreate→onStart→onResume。日志如下：

```
11:30:18.352: D/ActJumpActivity(2315): onCreate
11:30:18.352: D/ActJumpActivity(2315): onStart
11:30:18.352: D/ActJumpActivity(2315): onResume
```

从 ActJumpActivity 跳转到 ActNextActivity，调用方法的顺序为：上一个页面 onPause→下

一个页面 onCreate→onStart→onResume→上一个页面 onStop。日志如下：

```
11:30:32.668：D/ActJumpActivity(2315)：onPause
11:30:32.688：D/ActNextActivity(2315)：onCreate
11:30:32.688：D/ActNextActivity(2315)：onStart
11:30:32.688：D/ActNextActivity(2315)：onResume
11:30:33.116：D/ActJumpActivity(2315)：onStop
```

从 ActNextActivity 回到 ActJumpActivity（按返回键或在代码中调用 finish 方法），调用的方法顺序为：下一个页面 onPause→上一个页面 onRestart→onStart→onResume→下一个页面 onStop→onDestroy。日志如下：

```
11:30:40.740：D/ActNextActivity(2315)：onPause
11:30:40.752：D/ActJumpActivity(2315)：onRestart
11:30:40.752：D/ActJumpActivity(2315)：onStart
11:30:40.752：D/ActJumpActivity(2315)：onResume
11:30:41.160：D/ActNextActivity(2315)：onStop
11:30:41.164：D/ActNextActivity(2315)：onDestroy
```

至此，基本上可以弄清楚页面跳转时的生命周期了。总体上是跳转前的页面先调用 onPause 方法，然后跳转后的页面依次调用 onCreate/onRestart→onStart→onResume，最后跳转前的页面调用 onStop 方法（若返回上级页面，则下级页面还需调用 onDestroy 方法）。

2. 竖屏与横屏的切换

首先进入测试页面 ActRotateActivity，此时默认为竖屏显示；接着倒转手机切换到横屏，观察日志；然后倒转手机切换回竖屏，观察日志。3 个屏幕的显示日志时间没有重复，这里的日志截图是 3 次截图拼接而成的，如图 3-21 所示。



图 3-21 活动页面在横竖屏切换时的界面日志截图

从日志截图可以看出，竖屏与横屏似乎在每次切换时页面都要重新创建。为进一步验证实验结果，再一次查看 logcat 里的日志，日志信息如下：

```
21:02:10.179 D/ActRotateActivity：onCreate
21:02:10.179 D/ActRotateActivity：onStart
21:02:10.179 D/ActRotateActivity：onResume
21:02:13.227 D/ActRotateActivity：onPause
21:02:13.227 D/ActRotateActivity：onStop
21:02:13.227 D/ActRotateActivity：onDestroy
21:02:13.247 D/ActRotateActivity：onCreate
21:02:13.247 D/ActRotateActivity：onStart
21:02:13.247 D/ActRotateActivity：onResume
```

```
21:02:16.239 D/ActRotateActivity：onPause
21:02:16.239 D/ActRotateActivity：onStop
21:02:16.239 D/ActRotateActivity：onDestroy
21:02:16.279 D/ActRotateActivity：onCreate
21:02:16.279 D/ActRotateActivity：onStart
21:02:16.279 D/ActRotateActivity：onResume
```

分析日志的时间与内容，无论是竖屏切换到横屏，还是横屏切换到竖屏，都是原屏幕的页面从 onPause 到 onStop 再到 onDestroy 一路销毁，然后新屏幕的页面从 onCreate 到 onStart 再到 onResume 一路创建而来。

3. 按 HOME 键与返回 App

首先进入测试页面 ActHomeActivity；接着按 HOME 键，屏幕回到桌面；然后按任务键或长按 HOME 键（不同手机的操作不一样），屏幕调出进程视图；最后点击测试 App，屏幕返回测试页面。一路下来的屏幕日志截图如图 3-22 所示。

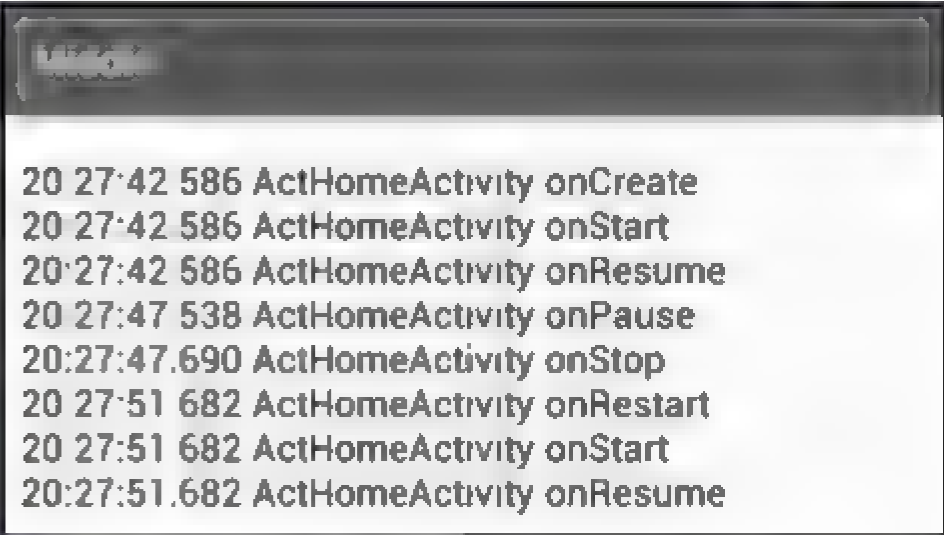


图 3-22　按 HOME 键的界面日志截图

从日志截图可以看到，此时测试页面的生命周期是典型的从活动状态变为暂停状态（回到桌面时）再到活动状态（返回 App 页面时）。观察 logcat 的后台日志，发现后台日志与屏幕日志保持一致。

3.5.2　使用 Intent 传递消息

Intent 的中文名是意图，意思是我想让你干什么，简单地说，就是传递消息。Intent 是各个组件之间信息沟通的桥梁，既能在 Activity 之间沟通，又能在 Activity 与 Service 之间沟通，也能在 Activity 与 Broadcast 之间沟通。总而言之，Intent 用于处理 Android 各组件之间的通信，完成的工作主要有 3 部分：

（1）Intent 需标明本次通信请求从哪里来、到哪里去、要怎么走。

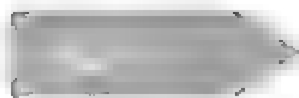
（2）发起方携带本次通信需要的数据内容，接收方对收到的 Intent 数据进行解包。

（3）如果发起方要求判断接收方的处理结果，Intent 就要负责让接收方传回应答的数据内容。

为了做好以上工作，就要给 Intent 配上必须的装备，Intent 的组成部分见表 3-5。

表 3-5　Intent 组成元素的列表说明

元素名称	设置方法	说明与用途
Component	setComponent	组件，用于指定 Intent 的来源与目的
Action	setAction	动作，用于指定 Intent 的操作行为
Data	setData	即 Uri，用于指定动作要操纵的数据路径
Category	addCategory	类别，用于指定 Intent 的操作类别



（续表）

元素名称	设置方法	说明与用途
Type	setType	数据类型，用于指定 Data 类型的定义
Extras	putExtra	扩展信息，用于指定装载的参数信息
Flags	setFlags	标志位，用于指定 Intent 的运行模式（启动标志）

表达 Intent 的来往路径有两种方式，一种是显式 Intent，另一种是隐式 Intent。

1. 显式 Intent，直接指定来源类与目标类名，属于精确匹配。

在声明一个 Intent 对象时，需要指定两个参数，第一个参数表示跳转的来源页面，第二个参数表示接下来要跳转到的页面类。具体的声明方式有如下 3 种：

（1）在构造函数中指定，示例代码如下：

```
Intent intent = new Intent(this, ActResponseActivity.class);
```

（2）调用 setClass 方法指定，示例代码如下：

```
Intent intent = new Intent();
intent.setClass(this, ActResponseActivity.class);
```

（3）调用 setComponent 方法指定，示例代码如下：

```
Intent intent = new Intent();
ComponentName component = new ComponentName(this, ActResponseActivity.class);
intent.setComponent(component);
```

2. 隐式 Intent，没有明确指定要跳转的类名，只给出一个动作让系统匹配拥有相同字串定义的目标，属于模糊匹配。

因为我们常常不希望直接暴露源码的类名，只给出一个事先定义好的名称，这样大家约定俗成、按图索骥就好，所以隐式 Intent 起到了过滤作用。这个定义好的动作名称是一个字符串，可以是自己定义的动作，也可以是已有的系统动作。系统动作的取值说明见表 3-6。

表 3-6　系统动作的取值说明

Intent 类的系统动作常量名	系统动作的常量值	说明
ACTION_MAIN	android.intent.action.MAIN	App 启动时的入口
ACTION_VIEW	android.intent.action.VIEW	显示数据给用户
ACTION_EDIT	android.intent.action.EDIT	显示可编辑的数据
ACTION_CALL	android.intent.action.CALL	拨号
ACITON_DIAL	android.intent.action.DIAL	打电话
ACTION_SEND	android.intent.action.SEND	发短信
ACTION_ANSWER	android.intent.action.ANSWER	接电话
ACTION_SEARCH	android.intent.action.SEARCH	导航栏上 SearchView 的搜索动作



这个动作名称通过 setAction 方法指定，也可以通过构造函数 Intent(String action)直接生成 Intent 对象。当然，由于动作是模糊匹配，因此有时需要更详细的路径，比如知道某人住在天通苑小区，并不能直接找到他家，还得说明他住在天通苑的哪一期、哪号楼、哪一层、哪一个单元。Uri 和 Category 便是这样的路径与门类信息，Uri 数据可通过构造函数 Intent(String action, Uri uri)在生成对象时一起指定，也可通过 setData 方法指定（setData 这个名字有歧义，实际就是 setUri）；Category 可通过 addCategory 方法指定，之所以用 add 而不用 set 方法，是因为一个 Intent 可同时设置多个 Category，一起进行过滤。

下面是一个调用系统拨号程序的例子，其中就用到了 Uri：

```
Intent intent = new Intent();
intent.setAction(Intent.ACTION_CALL);
Uri uri = Uri.parse("tel:"+"15960238696");
intent.setData(uri);
startActivity(intent);
```

隐式 Intent 还用到了过滤器的概念，即把不符合匹配条件的过滤掉，剩下符合条件的按照优先顺序调用。创建一个 Android 工程，AndroidManifest.xml 里的 intent-filter 就是 XML 中的过滤器。比如下面这个最常见的主页面 MainAcitivity，activity 节点下面便设置了 action 和 category 的过滤条件。其中，android.intent.action.MAIN 表示 App 的入口动作，android.intent.category.LAUNCHER 表示在 App 启动时调用。

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

3.5.3 向下一个 Activity 传递参数

前面说了，Intent 的 setData 方法只指定到达目标的路径，并非本次通信所携带的参数信息，真正的参数信息存放在 Extras 中。Intent 重载了很多种 putExtra 方法传递各种类型的参数，包括 String、int、double 等基本数据类型，甚至 Parcelable、Serializable 等序列化结构。不过只是调用 putExtra 方法显然不好管理，像送快递一样大小包裹随便扔，不但找起来不方便，丢了也难以知道。所以 Android 引入了 Bundle 概念，我们可以把 Bundle 理解为超市的寄包柜或快递收件柜，大小包裹由 Bundle 统一存取，方便又安全。

Bundle 内部用于存放数据的实质结构是 Map 映射，可添加元素、删除元素，还可判断元素是否存在。开发者把 Bundle 全部打包好只需调用一次 putExtras 方法，把 Bundle 全部取出来也只需调用一次 getExtras 方法。

下面是前一个页面向后一个页面发送请求数据的代码：



```
Intent intent = new Intent(MainActivity.this, FirstActivity.class);
Bundle bundle = new Bundle();
bundle.putString("name", "张三");
bundle.putInt("age", 30);
bundle.putDouble("height", 170.0f);
intent.putExtras(bundle);
startActivity(intent);
```

下面是后一个页面接收前一个页面请求数据的代码：

```
Intent intent = getIntent();
Bundle bundle = intent.getExtras();
String name = bundle.getString("name", "");
int age = bundle.getInt("age", 0);
double height = bundle.getDouble("height", 0.0f);
```

3.5.4　向上一个 Activity 返回参数

如同一般的通信一样，Intent 有时只把请求数据发送到下一个页面就行，有时还要处理下一个页面的应答数据（通常发生在下一个页面返回到上一个页面时）。如果只把请求数据发送到下一个页面，前一个页面调用 startActivity 方法就可以；如果还要处理下个页面的应答数据，此时就得分多步处理，详细步骤如下：

步骤01 前一个页面打包好请求数据，调用方法 startActivityForResult(Intent intent, int requestCode)，表示需要处理结果数据，第一个参数表示请求编号，用于标识每次请求的唯一性。

步骤02 后一个页面接收请求数据，进行相应处理。

步骤03 后一个页面在返回前一个页面时，打包应答数据并调用 setResult 方法返回信息。setResult 的第一个参数表示应答代码（成功还是失败），代码示例如下：

```
Intent intent = new Intent();
Bundle bundle = new Bundle();
bundle.putString("job", "码农");
intent.putExtras(bundle);
setResult(Activity.RESULT_OK, intent);
finish(); //表示关闭当前页面
```

步骤04 前一个页面重写方法 onActivityResult，该方法的输入参数包含请求编号和应答代码，请求编号用于判断对应哪次请求，应答代码用于判断后一个页面是否处理成功。然后对应答数据进行解包处理，代码示例如下：

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent intent) {
    Log.d(TAG, "onActivityResult. requestCode="+requestCode+", resultCode="+resultCode);
    Bundle resp = intent.getExtras();
    String job = resp.getString("job");
```

```
            Toast.makeText(this, "您目前的职业是"+job, Toast.LENGTH_LONG).show();
        }
```

下面是完整的请求页面代码与应答页面代码，结合效果界面加深对 Activity 处理参数传递的理解。请求页面的代码如下：

```
public class ActRequestActivity extends AppCompatActivity implements OnClickListener {
    private EditText et_request;
    private TextView tv_request;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_act_request);
        findViewById(R.id.btn_act_request).setOnClickListener(this);
        et_request = (EditText) findViewById(R.id.et_request);
        tv_request = (TextView) findViewById(R.id.tv_request);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_act_request) {
            Intent intent = new Intent();
            intent.setClass(this, ActResponseActivity.class);
            intent.putExtra("request_time", DateUtil.getNowTime());
            intent.putExtra("request_content", et_request.getText().toString());
            startActivityForResult(intent, 0);
        }
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (data != null) {
            String response_time = data.getStringExtra("response_time");
            String response_content = data.getStringExtra("response_content");
            String desc = String.format("收到返回消息 ：\n 应答时间为%s\n 应答内容为%s",
                    response_time, response_content);
            tv_request.setText(desc);
        }
    }
}
```



应答页面的代码如下：

```
public class ActResponseActivity extends AppCompatActivity implements OnClickListener {
    private EditText et_response;
    private TextView tv_response;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_act_response);
        findViewById(R.id.btn_act_response).setOnClickListener(this);
        et_response = (EditText) findViewById(R.id.et_response);
        tv_response = (TextView) findViewById(R.id.tv_response);
        Bundle bundle = getIntent().getExtras();
        String request_time = bundle.getString("request_time");
        String request_content = bundle.getString("request_content");
        String desc = String.format("收到请求消息 ：\n 请求时间为%s\n 请求内容为%s",
                request_time, request_content);
        tv_response.setText(desc);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_act_response) {
            Intent intent = new Intent();
            Bundle bundle = new Bundle();
            bundle.putString("response_time", DateUtil.getNowTime());
            bundle.putString("response_content", et_response.getText().toString());
            intent.putExtras(bundle);
            setResult(Activity.RESULT_OK, intent);
            finish();
        }
    }
}
```

具体的效果图分别如图 3-23、图 3-24、图 3-25 所示。其中，图 3-23 是当前页面要向下一个页面发送请求时的界面，图 3-24 是下一个页面准备返回上一个页面时的界面，图 3-25 是上一个页面收到下一个页面应答时的界面。



图 3-23　准备向下一个页面发送请求

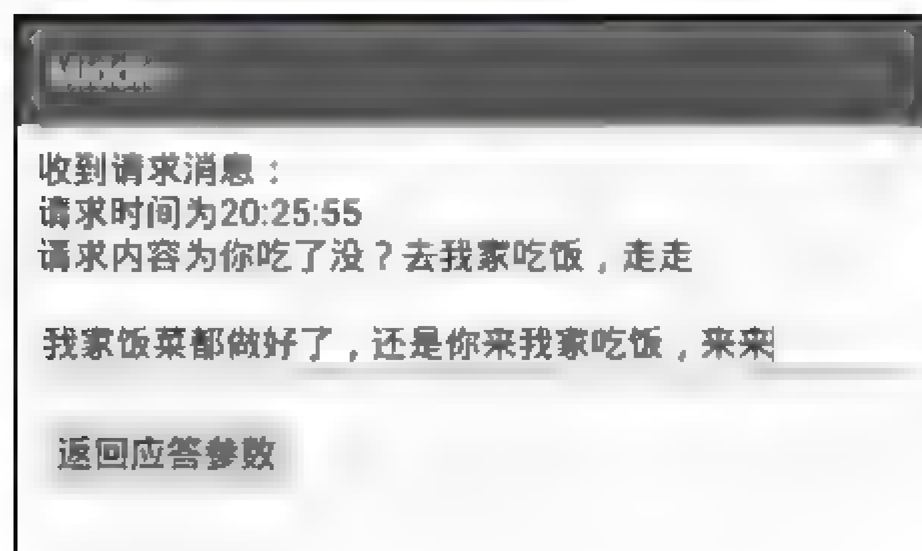


图 3-24　下一个页面准备返回消息



图 3-25　上一个页面收到返回消息

3.6　实战项目：登录 App

现在是实战项目时间，大家如此费力地看书学习，还不就是为了在实际项目中派上用场。凡是赚钱的 App，都要掌握用户资源，这便少不了为用户提供登录页面。下面设计并实现 App 的登录功能。

3.6.1　设计思路

如今楼市疯狂上涨，要买房自然少不了房贷，根据不同的贷款方式与还款方式计算出的月供数额各不相同。如果手机上有房贷计算器，就会便利许多。房贷计算器绝对是一个方便实用的 App，本书迄今为止介绍的 App 开发知识足够写一个房贷计算器 App 了，如图 3-26 所示。本章学到的主要控件基本都能派上用场，包括 RelativeLayout、EditText、RadioButton、CheckBox、Spinner 等。读者若有兴趣可自行编码练习，补充房贷计算的具体业务逻辑。

本章的实战项目最终选定 App 登录页面，是因为要复习 Activity 的相关概念与用法。Activity 是 Android 中最常用的组件，后续章节全部都会用到，所以要好好加以巩固。

各家 App 的登录页面大同小异，要么是用户名与密码组合登录，要么是手机号与验证码组合登录，如果要做得更好一点，就要提供忘记密码与记住密码等功能。我们的 App 登录项目把这些功能综合一下，都呈现到页面上，因为是练手，所以尽量让学到的控件都派上用场。登录页面的设计图初稿如图 3-27 所示。

读者找找看这个效果图包含哪些本章的新控件？一定会发现以下 6 个控件。

- 单选按钮 RadioButton：用来区分是密码登录还是验证码登录。
- 下拉框 Spinner：用于区分用户类型是个人用户还是公司用户。
- 编辑框 EditText：用来输入手机号码和密码。
- 复选框 CheckBox：用于判断是否记住密码。
- 相对布局 RelativeLayout：指定手机号码的编辑框放在手机号码 TextView 的右边。这里使用线性布局 LinearLayout 也可以。
- 框架布局 FrameLayout：忘记密码的按钮与密码输入框叠加。



图 3-26　房贷计算器的效果图



图 3-27　登录页面的效果图

至此，本章介绍的新控件基本都派上用场了。另外，本项目还要演示活动页面的的跳转功能，点击“忘记密码”按钮跳转到找回密码页面，找回密码页面的效果如图 3-28 所示。

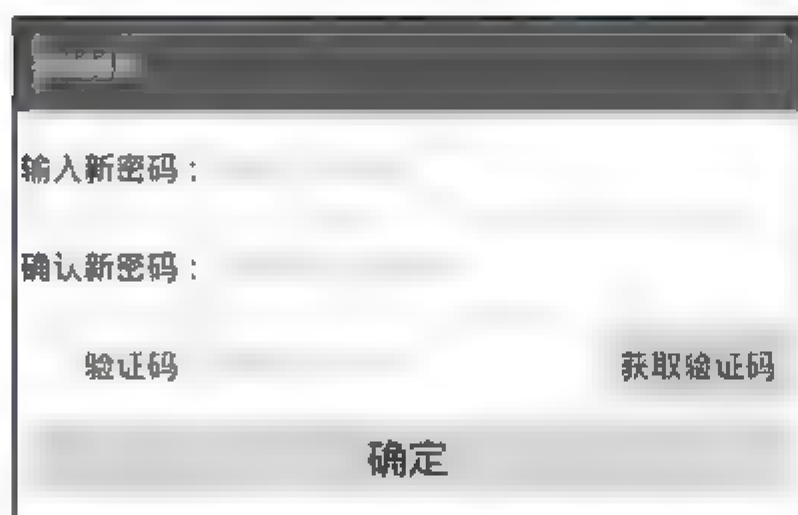


图 3-28　找回密码页面的效果图

找回密码的页面挺简单，主要问题是两个页面之间的跳转有哪些注意事项？页面跳转肯定要传递参数，一般唯一标识的手机号码要传过去，不然下一个页面不知道要为哪个手机号码修改密码；新密码也要传回去，不然上一个页面不知道密码被改成什么了。

另外，有一个细微的用户体验问题：用户会去找回密码，肯定是发现输入的密码不对。修改完密码回到登录页面时，密码输入框里还是原来错误的密码，此时用户清空错误密码才能输入新密码。我们的 App 想让用户觉得好用，就得急用户之所急、想用户之所想，像之前错误密码的情况应当由 App 在返回登录页面时自动清空原来错误的密码。自动清空的操作放在 onActivityResult 方法中处理是一个办法，但这样处理有一个问题，如果用户直接按返回键回到登录页面，onActivityResult 方法发现数据为空就不会处理。

这个问题其实不难，只要认真看书，结合前面关于 Activity 生命周期的说明，就能够找到解决办法。重写 onRestart 方法（确保是返回页面），在方法内部加上清空密码框的处理即可。这样一来，无论用户是修改完密码回到登录页，还是点击返回键回到登录页，App 都会自动清空密码框。

3.6.2　小知识：提醒对话框 AlertDialog

使用验证码登录时，App 要向用户手机发送短信验证码，但发送短信需要服务器支持，所以这里暂时使用随机数模拟验证码，然后以对话框的形式在界面上提示用户。另外，在登录的过程中，App 时常需要弹窗提示用户选择“是”或“否”，以此判断下一步的处理逻辑。在本实战项目开始之前，建议读者先演练一下提醒对话框（AlertDialog）的用法。

AlertDialog 是 Android 中最常用的对话框，可以完成常见的交互操作，如提示、确认、选择等功能。AlertDialog 没有公开的构造函数，必须借助 AlertDialog.Builder 才能完成参数设置，AlertDialog.Builder 的常用方法如下。

- setIcon：设置标题的图标。
- setTitle：设置标题的文本。
- setMessage：设置内容的文本。
- setPositiveButton：设置肯定按钮的信息，包括按钮文本和点击监听器。
- setNegativeButton：设置否定按钮的信息，包括按钮文本和点击监听器。
- setNeutralButton：设置中性按钮的信息，包括按钮文本和点击监听器，该方法比较少用。

通过 AlertDialog.Builder 设置完参数，还需调用 create 方法才能生成 AlertDialog 对象。最后调用 AlertDialog 对象的 show 方法，在页面上弹出提醒对话框。

下面是个提醒对话框的代码：

```
public void onClick(View v) {
    if (v.getId() == R.id.btn_alert) {
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("尊敬的用户");
        builder.setMessage("你真的要卸载我吗？");
        builder.setPositiveButton("残忍卸载", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                tv_alert.setText("虽然依依不舍，但是只能离开了");
            }
        });
        builder.setNegativeButton("我再想想", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                tv_alert.setText("让我再陪你三百六十五个日夜");
            }
        });
        AlertDialog alert = builder.create();
        alert.show();
    }
}
```

提醒对话框的弹窗效果如图 3-29 所示，该对话框有标题、有内容，还有两个按钮。

用户点击不同的按钮会触发不同的处理逻辑。图 3-30 所示为点击“我再想想”按钮后的页面，图 3-31 所示为点击“残忍卸载”按钮后的页面。

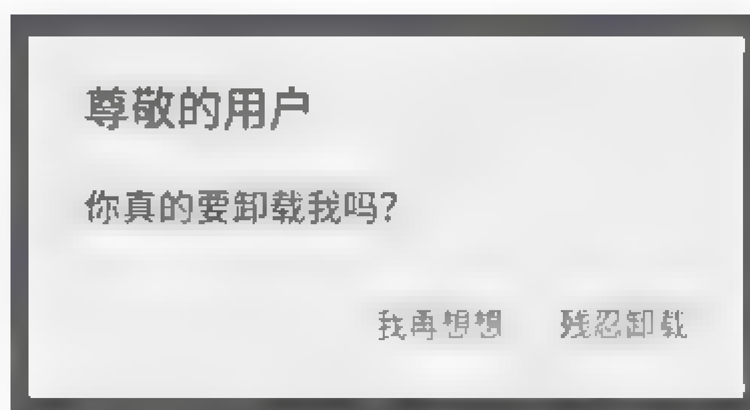


图 3-29　AlertDialog 的效果图



图 3-30　点击“我再想想”的截图



图 3-31　点击“残忍卸载”的截图

3.6.3　代码示例

前面的设计不但给出了两个页面的效果图，而且给出了业务逻辑的大概思路，接下来主要是编码将其实现。编码过程分为 3 个步骤：

01 先想好代码文件与布局文件的名称，比如登录页面的代码文件取名 LoginMainActivity.java，布局文件取名 activity_login.xml；找回密码页面的代码文件取名 LoginForgetActivity.java，布局文件取名 activity_login_forget.xml。记得在 AndroidManifest.xml 中注册两个页面的 acitivity 节点，注册代码如下：

```
<activity android:name=".LoginMainActivity" />
<activity android:name=".LoginForgetActivity" />
```

02 在 res/layout 目录下创建布局文件 activity_login.xml 和 activity_login_forget.xml，根据页面效果图编写两个页面的布局定义文件。

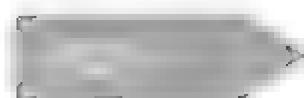
03 在项目的包名目录下创建类 LoginMainActivity 和 LoginForgetActivity，填入具体的控件操作与业务逻辑代码。

下面是登录页面 LoginMainActivity.java 的主要代码片段：

```
public void onClick(View v) {
    String phone = et_phone.getText().toString();
    if (v.getId() == R.id.btn_forget) {
        if (phone==null || phone.length()<11) {
            Toast.makeText(this, "请输入正确的手机号", Toast.LENGTH_SHORT).show();
            return;
        }
        if (rb_password.isChecked() == true) {
            Intent intent = new Intent(this, LoginForgetActivity.class);
            intent.putExtra("phone", phone);
            startActivityForResult(intent, mRequestCode);
        } else if (rb_verifycode.isChecked() == true) {
            mVerifyCode = String.format("%06d", (int)(Math.random()*1000000%1000000));
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("请记住验证码");
            builder.setMessage("手机号"+phone+"，本次验证码是"+mVerifyCode+"，请输入验证码");
            builder.setPositiveButton("好的", null);
            AlertDialog alert = builder.create();
            alert.show();
```

```
                }
            } else if (v.getId() == R.id.btn_login) {
                if (phone==null || phone.length()<11) {
                    Toast.makeText(this, "请输入正确的手机号", Toast.LENGTH_SHORT).show();
                    return;
                }
                if (rb_password.isChecked() == true) {
                    if (et_password.getText().toString().equals(mPassword) != true) {
                        Toast.makeText(this, "请输入正确的密码", Toast.LENGTH_SHORT).show();
                        return;
                    } else {
                        loginSuccess();
                    }
                } else if (rb_verifycode.isChecked() == true) {
                    if (et_password.getText().toString().equals(mVerifyCode) != true) {
                        Toast.makeText(this, "请输入正确的验证码", Toast.LENGTH_SHORT).show();
                        return;
                    } else {
                        loginSuccess();
                    }
                }
            }
        }
        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            if (requestCode == mRequestCode && data!=null) {
                //用户密码已改为新密码
                mPassword = data.getStringExtra("new_password");
            }
        }
        //从修改密码页面返回登录页面，要清空密码的输入框
        @Override
        protected void onRestart() {
            et_password.setText("");
            super. onRestart();
        }

        private void loginSuccess() {
            String desc = String.format("您的手机号码是%s，类型是%s。恭喜你通过登录验证，点击“确
定”按钮返回上个页面", et_phone.getText().toString(), typeArray[mType]);
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("登录成功");
```



```
        builder.setMessage(desc);
        builder.setPositiveButton("确定返回", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                finish();
            }
        });
        builder.setNegativeButton("我再看看", null);
        AlertDialog alert = builder.create();
        alert.show();
    }
```

下面是找回密码页面 LoginForgetActivity.java 的代码：

```
public class LoginForgetActivity extends AppCompatActivity implements OnClickListener {
    private EditText et_password_first;
    private EditText et_password_second;
    private EditText et_verifycode;
    private String mVerifyCode;
    private String mPhone;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login_forget);
        et_password_first = (EditText) findViewById(R.id.et_password_first);
        et_password_second = (EditText) findViewById(R.id.et_password_second);
        et_verifycode = (EditText) findViewById(R.id.et_verifycode);
        findViewById(R.id.btn_verifycode).setOnClickListener(this);
        findViewById(R.id.btn_confirm).setOnClickListener(this);
        mPhone = getIntent().getStringExtra("phone");
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_verifycode) {
            if (mPhone==null || mPhone.length()<11) {
                Toast.makeText(this, "请输入正确的手机号", Toast.LENGTH_SHORT).show();
                return;
            }
            mVerifyCode = String.format("%06d", (int) (Math.random() * 1000000 % 1000000));
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("请记住验证码");
            builder.setMessage("手机号"+mPhone+"，本次验证码是"+mVerifyCode+"，请输入验证码");
            builder.setPositiveButton("好的", null);
```

```
                AlertDialog alert = builder.create();
                alert.show();
            } else if (v.getId() == R.id.btn_confirm) {
                String password_first = et_password_first.getText().toString();
                String password_second = et_password_second.getText().toString();
                if (password_first==null || password_first.length()<6 ||
                        password_second==null || password_second.length()<6) {
                    Toast.makeText(this, "请输入正确的新密码", Toast.LENGTH_SHORT).show();
                    return;
                } else if (password_first.equals(password_second) != true) {
                    Toast.makeText(this, "两次输入的新密码不一致", Toast.LENGTH_SHORT).show();
                    return;
                } else if (et_verifycode.getText().toString().equals(mVerifyCode) != true) {
                    Toast.makeText(this, "请输入正确的验证码", Toast.LENGTH_SHORT).show();
                    return;
                } else {
                    Toast.makeText(this, "密码修改成功", Toast.LENGTH_SHORT).show();
                    Intent intent = new Intent();
                    intent.putExtra("new_password", password_first);
                    setResult(Activity.RESULT_OK, intent);
                    finish();
                }
            }
        }
    }
```

3.7 小　　结

本章主要介绍 App 开发的中级控件相关知识，包括其他布局的用法（相对布局、框架布局）、特殊按钮的用法（复选框、开关按钮、单选按钮）、适配视图的基本用法（下拉框、数组适配器、简单适配器）、编辑框的用法（文本编辑框、自动完成编辑框）、Activity 组件的基本用法（生命周期、意图、传递消息）。最后设计了一个实战项目“登录 App”，在该项目的 App 编码中，采用前面介绍的大部分布局和控件，以及 Activity 跳转与返回时的消息请求与应答，初步在实际代码中运用生命周期方法。最后介绍了提醒对话框的用法。

通过本章的学习，读者应该能掌握以下 3 种开发技能：

（1）在布局文件中合理使用本章学到的布局和控件。

（2）在代码中合理调用本章学到的布局和控件的相关方法。

（3）学会 Activity 组件的用法，如在页面之间跳转的消息传递操作和在合适的场合重写生命周期的方法。





数据存储

本章介绍 Android 四种主要存储方式的用法，包括共享参数 SharedPreferences、数据库 SQLite、SD 卡文件、App 的全局内存，另外介绍重要组件之一的 Application 的基本概念与常见用法。最后，结合本章所学的知识演示实战项目“购物车”的设计与实现。

4.1 共享参数 SharedPreferences

本节介绍 Android 的键值对存储方式——共享参数 SharedPreferences 的使用方法，包括如何保存数据与读取数据，通过共享参数结合“登录 App”项目实现记住密码功能。

4.1.1 基本用法

SharedPreferences 是 Android 的一个轻量级存储工具，采用的存储结构是 Key-Value 的键值对方式，类似于 Java 的 Properties 类，二者都是把 Key-Value 的键值对保存在配置文件中。不同的是 Properties 的文件内容是 Key=Value 这样的形式，而 SharedPreferences 的存储介质是符合 XML 规范的配置文件。保存 SharedPreferences 键值对信息的文件路径是/data/data/应用包名/shared_prefs/文件名.xml。下面是一个共享参数的 XML 文件示例：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="name">Mr Lee</string>
    <int name="age" value="30" />
    <boolean name="married" value="true" />
    <float name="weight" value="100.0" />
</map>
```

基于 XML 格式的特点，SharedPreferences 主要适用于如下场合：

（1）简单且孤立的数据。若是复杂且相互间有关的数据，则要保存在数据库中。

（2）文本形式的数据。若是二进制数据，则要保存在文件中。

（3）需要持久化存储的数据。在 App 退出后再次启动时，之前保存的数据仍然有效。

实际开发中，共享参数经常存储的数据有 App 的个性化配置信息、用户使用 App 的行为信息、临时需要保存的片段信息等。

SharedPreferences 对数据的存储和读取操作类似于 Map，也有 put 函数用于存储数据、get 函数用于读取数据。在使用共享参数之前，要先调用 getSharedPreferences 函数声明文件名与操作模式，示例代码如下：

```
SharedPreferences sps= getSharedPreferences("share", Context.MODE_PRIVATE);
```

getSharedPreferences 方法的第一个参数是文件名，上面的 share 表示当前使用的共享参数文件名是 share.xml；第二个参数是操作模式，一般都填 MODE_PRIVATE，表示私有模式。

共享参数存储数据要借助于 Editor 类，示例代码如下：

```
SharedPreferences.Editor editor = sps.edit();
editor.putString("name", "Mr Lee");
editor.putInt("age", 30);
editor.putBoolean("married", true);
```

```
editor.putFloat("weight", 100f);
editor.commit();
```

共享参数读取数据相对简单，直接使用对象即可完成数据读取方法的调用，注意 get 方法的第二个参数表示默认值，示例代码如下：

```
String name = sps.getString("name", "");
int age = sps.getInt("age", 0);
boolean married = sps.getBoolean("married", false);
float weight = sps.getFloat("weight", 0);
```

下面通过页面录入信息演示 SharedPreferences 的存取过程，如图 4-1 所示。在页面上利用 EditText 录入用户注册信息，并保存到共享参数文件中。在另一个页面，App 从共享参数文件中读取用户注册信息，并将注册信息依次显示在页面中，如图 4-2 所示。



图 4-1　写入共享参数



图 4-2　从共享参数读取

4.1.2　实现记住密码功能

上一章的实战项目“登录 App”页面下方有一个“记住密码”复选框，当时只是为了演示控件的运用，并未真正记住密码。因为用户退出后重新进入登录页面，App 没有回忆起上次用户的登录密码。现在我们利用共享参数对该项目进行改造，使之实现记住密码的功能。

改造的内容主要有 3 处：

（1）声明一个 SharedPreferences 对象，并在 onCreate 函数中调用 getSharedPreferences 方法对该对象进行初始化操作。

（2）登录成功时，如果用户勾选了“记住密码”，就使用共享参数保存手机号码与密码。在 loginSuccess 函数中增加如下代码：

```
if (bRemember) {
    SharedPreferences.Editor editor = mShared.edit();
    editor.putString("phone", et_phone.getText().toString());
    editor.putString("password", et_password.getText().toString());
    editor.commit();
}
```

（3）在打开登录页面时，App 从共享参数中读取手机号码与密码，并展示在界面上。在

onCreate 函数中增加如下代码：

```
String phone = mShared.getString("phone", "");
String password = mShared.getString("password", "");
et_phone.setText(phone);
et_password.setText(password);
```

修改完毕后，如果不出意料，只要用户上次登录成功时勾选"记住密码"，下次进入登录页面时 App 就会自动填写上次登录的手机号码与密码。具体的效果如图 4-3 和图 4-4 所示。其中，图 4-3 所示为用户首次登录成功，此时勾选了"记住密码"；图 4-4 所示为用户再次进入登录页面，因为上次登录成功时已经记住密码，所以这次页面会自动展示保存的登录信息。



图 4-3　将登录信息保存到共享参数



图 4-4　从共享参数读取登录信息

4.2　数据库 SQLite

本节介绍 Android 的数据库存储方式—— SQLite 的使用方法，包括如何建表和删表、变更表结构以及对表数据进行增加、删除、修改、查询等操作，然后通过 SQLite 结合"登录 App"项目改进记住密码功能。

4.2.1　SQLite 的基本用法

SQLite 是一个小巧的嵌入式数据库，使用方便、开发简单，手机上最早由 iOS 运用，后来 Android 也采用了 SQLite。SQLite 的多数 sql 语法与 Oracle 一样，下面只列出不同的地方：

（1）建表时为避免重复操作，应加上 IF NOT EXISTS 关键词，例如 CREATE TABLE IF NOT EXISTS table_name。

（2）删表时为避免重复操作，应加上 IF EXISTS 关键词，例如 DROP TABLE IF EXISTS table_name。

（3）添加新列时使用 ALTER TABLE table_name ADD COLUMN ...，注意比 Oracle 多了一个 COLUMN 关键字。

（4）在 SQLite 中，ALTER 语句每次只能添加一列，如果要添加多列，就只能分多次添加。



（5）SQLite 支持整型 INTEGER、字符串 VARCHAR、浮点数 FLOAT，但不支持布尔类型。布尔类型数要使用整型保存，如果直接保存布尔数据，在入库时 SQLite 就会自动将其转为 0 或 1，0 表示 false，1 表示 true。

（6）SQLite 建表时需要一个唯一标识字段，字段名为_id。每建一张新表都要例行公事加上该字段定义，具体属性定义为_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL。

（7）条件语句等号后面的字符串值要用单引号括起来，如果没用使用单引号括起来，在运行时就会报错。

SQLiteDatabase 是 SQLite 的数据库管理类，我们可以在活动页面代码或任何能取到 Context 的地方获取数据库实例，参考代码如下：

```
        //创建数据库，如果已存在就打开
        SQLiteDatabase db = getApplicationContext().openOrCreateDatabase("test.db",
Context.MODE_PRIVATE, null);
        //删除数据库
        getApplicationContext().deleteDatabase("test.db");
```

SQLiteDatabase 提供了若干操作数据表的 API，常用的方法有 3 类，列举如下：

1. 管理类，用于数据库层面的操作。

- openDatabase：打开指定路径的数据库。
- isOpen：判断数据库是否已打开。
- close：关闭数据库。
- getVersion：获取数据库的版本号。
- setVersion：设置数据库的版本号。

2. 事务类，用于事务层面的操作。

- beginTransaction：开始事务。
- setTransactionSuccessful：设置事务的成功标志。
- endTransaction：结束事务。执行本方法时，系统会判断是否已执行 setTransactionSuccessful，如果之前已设置就提交，如果没有设置就回滚。

3. 数据处理类，用于数据表层面的操作。

- execSQL：执行拼接好的 SQL 控制语句。一般用于建表、删表、变更表结构。
- delete：删除符合条件的记录。
- update：更新符合条件的记录。
- insert：插入一条记录。
- query：执行查询操作，返回结果集的游标。
- rawQuery：执行拼接好的 SQL 查询语句，返回结果集的游标。



4.2.2 SQLiteOpenHelper

SQLiteDatabase 存在局限性，例如必须小心、不能重复地打开数据库，处理数据库的升级很不方便。Android 提供了一个辅助工具—— SQLiteOpenHelper，用于指导我们进行 SQLite 的合理使用。

SQLiteOpenHelper 的具体使用步骤如下：

步骤 01 新建一个继承自 SQLiteOpenHelper 的数据库操作类，提示重写 onCreate 和 onUpgrade 两个方法。其中，onCreate 方法只在第一次打开数据库时执行，在此可进行表结构创建的操作；onUpgrade 方法在数据库版本升高时执行，因此我们可以在 onUpgrade 函数内部根据新旧版本号进行表结构变更处理。

步骤 02 封装保证数据库安全的必要方法，包括获取单例对象、打开数据库连接、关闭数据库连接。

- 获取单例对象：确保 App 运行时数据库只被打开一次，避免重复打开引起错误。
- 打开数据库连接：SQLite 有锁机制，即读锁和写锁的处理；故而数据库连接也分两种，读连接可调用 SQLiteOpenHelper 的 getReadableDatabase 方法获得，写连接可调用 getWritableDatabase 获得。
- 关闭数据库连接：数据库操作完毕后，应当调用 SQLiteDatabase 对象的 close 方法关闭连接。

步骤 03 提供对表记录进行增加、删除、修改、查询的操作方法。

可被 SQLite 直接使用的数据结构是 ContentValues 类，类似于映射 Map，提供 put 和 get 方法用来存取键值对。区别之处在于 ContentValues 的键只能是字符串，查看 ContentValues 的源码会发现其内部保存键值对的数据结构就是 HashMap “private HashMap<String, Object> mValues;”。ContentValues 主要用于记录增加和更新操作，即 SQLiteDatabase 的 insert 和 update 方法。

对于查询操作来说，使用的是另一个游标类 Cursor。调用 SQLiteDatabase 的 query 和 rawQuery 方法时，返回的都是 Cursor 对象，因此获取查询结果要根据游标的指示一条一条遍历结果集合。Cursor 的常用方法可分为 3 类，说明如下：

1. 游标控制类方法，用于指定游标的状态。

- close：关闭游标。
- isClosed：判断游标是否关闭。
- isFirst：判断游标是否在开头。
- isLast：判断游标是否在末尾。

2. 游标移动类方法，把游标移动到指定位置。

- moveToFirst：移动游标到开头。
- moveToLast：移动游标到末尾。
- moveToNext：移动游标到下一条记录。

- moveToPrevious：移动游标到上一条记录。
- move：往后移动游标若干条记录。
- moveToPosition：移动游标到指定位置的记录。

3. 获取记录类方法，可获取记录的数量、类型以及取值。

- getCount：获取结果记录的数量。
- getInt：获取指定字段的整型值。
- getFloat：获取指定字段的浮点数值。
- getString：获取指定字段的字符串值。
- getType：获取指定字段的字段类型。

鉴于数据库操作的特殊性，不方便单独演示某个功能，接下来从创建数据库开始介绍，完整演示一下数据库的读写操作。如图 4-5 和图 4-6 所示，在页面上分别录入两个用户的注册信息并保存到 SQLite。从 SQLite 读取用户注册信息并展示在页面上，如图 4-7 所示。

图 4-5　第一条注册信息保存到数据库

图 4-6　第二条注册信息保存到数据库

图 4-7　从 SQLite 中读取两条注册记录

下面是用户注册信息数据库的 SQLiteOpenHelper 操作类的完整代码：

```
public class UserDBHelper extends SQLiteOpenHelper {
    private static final String TAG = "UserDBHelper";
    private static final String DB_NAME = "user.db";
```

```
private static final int DB_VERSION = 1;
private static UserDBHelper mHelper = null;
private SQLiteDatabase mDB = null;
private static final String TABLE_NAME = "user_info";

private UserDBHelper(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
}

private UserDBHelper(Context context, int version) {
        super(context, DB_NAME, null, version);
}

public static UserDBHelper getInstance(Context context, int version) {
        if (version > 0 && mHelper == null) {
                mHelper = new UserDBHelper(context, version);
        } else if (mHelper == null) {
                mHelper = new UserDBHelper(context);
        }
        return mHelper;
}

public SQLiteDatabase openReadLink() {
        if (mDB == null || mDB.isOpen() != true) {
                mDB = mHelper.getReadableDatabase();
        }
        return mDB;
}

public SQLiteDatabase openWriteLink() {
        if (mDB == null || mDB.isOpen() != true) {
                mDB = mHelper.getWritableDatabase();
        }
        return mDB;
}

public void closeLink() {
        if (mDB != null && mDB.isOpen() == true) {
                mDB.close();
                mDB = null;
        }
}
```



```
    public String getDBName() {
        if (mHelper != null) {
            return mHelper.getDatabaseName();
        } else {
            return DB_NAME;
        }
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.d(TAG, "onCreate");
        String drop_sql = "DROP TABLE IF EXISTS " + TABLE_NAME + ";";
        db.execSQL(drop_sql);
        String create_sql = "CREATE TABLE IF NOT EXISTS " + TABLE_NAME + " ("
                + "_id INTEGER PRIMARY KEY  AUTOINCREMENT NOT NULL,"
                + "name VARCHAR NOT NULL," + "age INTEGER NOT NULL,"
                + "height LONG NOT NULL," + "weight FLOAT NOT NULL,"
                + "married INTEGER NOT NULL," + "update_time VARCHAR NOT NULL"
                //演示数据库升级时要先注释下面这行代码
                + ",phone VARCHAR" + ",password VARCHAR" + ");";
        Log.d(TAG, "create_sql:" + create_sql);
        db.execSQL(create_sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.d(TAG, "onUpgrade oldVersion="+oldVersion+", newVersion="+newVersion);
        if (newVersion > 1) {
            //Android 的 ALTER 命令不支持一次添加多列，只能分多次添加
            String alter_sql = "ALTER TABLE " + TABLE_NAME + " ADD COLUMN " + "phone
VARCHAR;";
            Log.d(TAG, "alter_sql:" + alter_sql);
            db.execSQL(alter_sql);
            alter_sql = "ALTER TABLE " + TABLE_NAME + " ADD COLUMN " + "password
VARCHAR;";
            Log.d(TAG, "alter_sql:" + alter_sql);
            db.execSQL(alter_sql);
        }
    }

    public int delete(String condition) {
        int count = mDB.delete(TABLE_NAME, condition, null);
        return count;
```

```
    }

    public int deleteAll() {
        int count = mDB.delete(TABLE_NAME, "1=1", null);
        return count;
    }

    public long insert(UserInfo info) {
        ArrayList<UserInfo> infoArray = new ArrayList<UserInfo>();
        infoArray.add(info);
        return insert(infoArray);
    }

    public long insert(ArrayList<UserInfo> infoArray) {
        long result = -1;
        for (int i = 0; i < infoArray.size(); i++) {
            UserInfo info = infoArray.get(i);
            ArrayList<UserInfo> tempArray = new ArrayList<UserInfo>();
            // 如果存在同名记录，就更新记录。注意条件语句的等号后面要用单引号括起来
            if (info.name!=null && info.name.length()>0) {
                String condition = String.format("name='%s'", info.name);
                tempArray = query(condition);
                if (tempArray.size() > 0) {
                    update(info, condition);
                    result = tempArray.get(0).rowid;
                    continue;
                }
            }
            // 如果存在同样的手机号码，就更新记录
            if (info.phone!=null && info.phone.length()>0) {
                String condition = String.format("phone='%s'", info.phone);
                tempArray = query(condition);
                if (tempArray.size() > 0) {
                    update(info, condition);
                    result = tempArray.get(0).rowid;
                    continue;
                }
            }
            // 如果不存在唯一性重复的记录，就插入新记录
            ContentValues cv = new ContentValues();
            cv.put("name", info.name);
            cv.put("age", info.age);
            cv.put("height", info.height);
```



```
                cv.put("weight", info.weight);
                cv.put("married", info.married);
                cv.put("update_time", info.update_time);
                cv.put("phone", info.phone);
                cv.put("password", info.password);
                result = mDB.insert(TABLE_NAME, "", cv);
                // 添加成功后返回行号，失败则返回-1
                if (result == -1) {
                    return result;
                }
            }
            return result;
        }

        public int update(UserInfo info, String condition) {
            ContentValues cv = new ContentValues();
            cv.put("name", info.name);
            cv.put("age", info.age);
            cv.put("height", info.height);
            cv.put("weight", info.weight);
            cv.put("married", info.married);
            cv.put("update_time", info.update_time);
            cv.put("phone", info.phone);
            cv.put("password", info.password);
            int count = mDB.update(TABLE_NAME, cv, condition, null);
            return count;
        }

        public int update(UserInfo info) {
            return update(info, "rowid="+info.rowid);
        }

        public ArrayList<UserInfo> query(String condition) {
            String sql = String.format("select rowid,_id,name,age,height,weight,married,update_time," +
                    "phone,password from %s where %s;", TABLE_NAME, condition);
            Log.d(TAG, "query sql："+sql);
            ArrayList<UserInfo> infoArray = new ArrayList<UserInfo>();
            Cursor cursor = mDB.rawQuery(sql, null);
            if (cursor.moveToFirst()) {
                for (;; cursor.moveToNext()) {
                    UserInfo info = new UserInfo();
                    info.rowid = cursor.getLong(0);
                    info.xuhao = cursor.getInt(1);
```

```
                    info.name = cursor.getString(2);
                    info.age = cursor.getInt(3);
                    info.height = cursor.getLong(4);
                    info.weight = cursor.getFloat(5);
                    //SQLite 没有布尔型，用 0 表示 false，用 1 表示 true
                    info.married = (cursor.getInt(6)==0)?false:true;
                    info.update_time = cursor.getString(7);
                    info.phone = cursor.getString(8);
                    info.password = cursor.getString(9);
                    infoArray.add(info);
                    if (cursor.isLast() == true) {
                        break;
                    }
                }
            }
            cursor.close();
            return infoArray;
        }

        public UserInfo queryByPhone(String phone) {
            UserInfo info = null;
            ArrayList<UserInfo> infoArray = query(String.format("phone='%s'", phone));
            if (infoArray.size() > 0) {
                info = infoArray.get(0);
            }
            return info;
        }
    }
```

4.2.3 优化记住密码功能

在“4.1.2 实现记住密码功能”中，我们利用共享参数实现了记住密码的功能，不过这个方法有局限，只能记住一个用户的登录信息，并且手机号码跟密码不存在从属关系，如果换个手机号码登录，前一个用户的登录信息就被覆盖了。真正意义上的记住密码功能是先输入手机号码，然后根据手机号匹配保存的密码，一个密码对应一个手机号码，从而实现具体手机号码的密码记忆功能。

现在我们运用 SQLite 技术分条存储不同用户的登录信息，并提供根据手机号码查找登录信息的方法，这样可以同时记住多个手机号码的密码。具体的改造主要有以下 3 点：

（1）声明一个 UserDBHelper 对象，然后在活动页面的 onResume 方法中打开数据库连接，在 onPasue 方法中关闭数据库连接，示例代码如下：



```
@Override
protected void onResume() {
    super.onResume();
    mHelper = UserDBHelper.getInstance(this, 2);
    mHelper.openWriteLink();
}

@Override
protected void onPause() {
    super.onPause();
    mHelper.closeLink();
}
```

（2）登录成功时，如果用户勾选了“记住密码”，就使用数据库保存手机号码与密码在内的登录信息。在 loginSuccess 函数中增加如下代码：

```
    if (bRemember) {
        UserInfo info = new UserInfo();
        info.phone = et_phone.getText().toString();
        info.password = et_password.getText().toString();
        info.update_time = DateUtil.getNowDateTime("yyyy-MM-dd HH:mm:ss");
        mHelper.insert(info);
    }
```

（3）再次打开登录页面，用户输入手机号完毕后点击密码输入框时，App 到数据库中根据手机号查找登录记录，并将记录结果中的密码填入密码框。

看到这里，读者也许已经想到给密码框注册点击事件，然后在 onClick 方法中补充数据库读取操作。可是 EditText 比较特殊，点击后只是让其获得焦点，再次点击才会触发点击事件。也就是说，要连续点击两次 EditText 才会处理点击事件。Android 有时就是这么调皮捣蛋，你让它往东，它偏偏往西。难不成叫用户将就一下点击两次？用户肯定觉得这个 App 古怪、难用，还是卸载好了……这里提供一个解决办法，先给密码框注册一个焦点变更监听器，比如下面这行代码：

```
    et_password.setOnFocusChangeListener(this);
```

这个焦点变更监听器要实现接口 OnFocusChangeListener，对应的事件处理方法是 onFocusChange，将数据库查询操作放在该方法中，详细代码如下：

```
@Override
public void onFocusChange(View v, boolean hasFocus) {
    String phone = et_phone.getText().toString();
    if (v.getId() == R.id.et_password) {
        if (phone.length() > 0 && hasFocus == true) {
            UserInfo info = mHelper.queryByPhone(phone);
            if (info != null) {
```

```
                        et_password.setText(info.password);
                    }
                }
            }
        }
```

这样，就不再需要点击两次才处理点击事件了。

代码写完后，再来看登录页面的效果图，用户上次登录成功时已勾选“记住密码”，现在再次进入登录页面，用户输入手机号后光标还停留在手机框，如图 4-8 所示。接着点击密码框，光标随之跳到密码框，这时密码框自动填入了该手机号对应的密码串，如图 4-9 所示。如此便真正实现了记住密码功能。



图 4-8　光标在手机号码框



图 4-9　光标在密码输入框

4.3　SD 卡文件操作

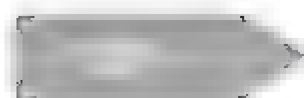
本节介绍 Android 的文件存储方式——SD 卡的用法，包括如何获取 SD 卡目录信息、在 SD 卡上读写文本文件、在 SD 卡读写图片文件等功能。

4.3.1　SD 卡的基本操作

手机的存储空间一般分为两块，一块用于内部存储，另一块用于外部存储（SD 卡）。早期的 SD 卡是可插拔式的存储芯片，不过自己买的 SD 卡质量参差不齐，经常会影响 App 的正常运行，所以后来越来越多手机把 SD 卡固化到手机内部，虽然拔不出来，但是 Android 仍然称之为外部存储。

获取手机上的 SD 卡信息通过 Environment 类实现，该类是 App 获取各种目录信息的工具，主要方法有以下 7 种。

- getRootDirectory：获得系统根目录的路径。
- getDataDirectory：获得系统数据目录的路径。
- getDownloadCacheDirectory：获得下载缓存目录的路径。



- getExternalStorageDirectory：获得外部存储（SD 卡）的路径。
- getExternalStorageState：获得 SD 卡的状态。

状态的具体取值说明见表 4-1。

表 4-1　SD 卡的存储状态取值说明

Environment 类的存储状态常量名	常量值	常量说明
MEDIA_UNKNOWN	unknown	未知
MEDIA_REMOVED	removed	已经移除
MEDIA_UNMOUNTED	unmounted	未挂载
MEDIA_CHECKING	checking	正在检查
MEDIA_NOFS	nofs	不支持的文件系统
MEDIA_MOUNTED	mounted	已经挂载，且是可读写状态
MEDIA_MOUNTED_READ_ONLY	mounted_ro	已经挂载，且是只读状态
MEDIA_SHARED	shared	当前未挂载，但通过 USB 共享
MEDIA_BAD_REMOVAL	bad_removal	未挂载就被移除
MEDIA_UNMOUNTABLE	unmountable	无法挂载
MEDIA_EJECTING	ejecting	正在弹出

- getStorageState：获得指定目录的状态。
- getExternalStoragePublicDirectory：获得 SD 卡指定类型目录的路径。

目录类型的具体取值说明见表 4-2。

表 4-2　SD 卡的目录类型取值说明

Environment 类的目录类型	常量值	常量说明
DIRECTORY_DCIM	DCIM	相片存放目录（包括相机拍摄的图片和视频）
DIRECTORY_DOCUMENTS	Documents	文档存放目录
DIRECTORY_DOWNLOADS	Download	下载文件存放目录
DIRECTORY_MOVIES	Movies	视频存放目录
DIRECTORY_MUSIC	Music	音乐存放目录
DIRECTORY_PICTURES	Pictures	图片存放目录

为正常操作 SD 卡，需要在 AndroidManifest.xml 中声明 SD 卡的权限，具体代码如下：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAG" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

下面演示一下 Environment 类各方法的使用效果，如图 4-10 所示。页面上展示了 Environment 类获取到的系统及 SD 卡的相关目录信息。





图 4-10　某设备上的 SD 卡目录信息

4.3.2　文本文件读写

文本文件的读写一般借助于 FileOutputStream 和 FileInputStream。其中，FileOutputStream 用于写文件，FileInputStream 用于读文件。文件输出输入流是 Java 语言的基础工具，这里不再赘述，直接给出具体的实现代码：

```
public static void saveText(String path, String txt) {
    try {
        FileOutputStream fos = new FileOutputStream(path);
        fos.write(txt.getBytes());
        fos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static String openText(String path) {
    String readStr = "";
    try {
        FileInputStream fis = new FileInputStream(path);
        byte[] b = new byte[fis.available()];
        fis.read(b);
        readStr = new String(b);
        fis.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return readStr;
}
```

文本文件的读写效果如图 4-11 所示。App 把页面录入的注册信息保存到 SD 卡的文本文

件中，接着进入文件列表读取页面，选中某个文本文件，页面就会展示该文件的文本内容，如图 4-12 所示。



图 4-11　将注册信息保存到文本文件



图 4-12　从文本文件读取注册信息

4.3.3　图片文件读写

Android 的图片处理类是 Bitmap，App 读写 Bitmap 可以使用 FileOutputStream 和 FileInputStream。不过在实际开发中，读写图片文件一般用性能更好的 BufferedOutputStream 和 BufferedInputStream。

保存图片文件时用到 Bitmap 的 compress 方法，可指定图片类型和压缩质量；打开图片文件时使用 BitmapFactory 的 decodeStream 方法。读写图片的具体代码如下：

```
public static void saveImage(String path, Bitmap bitmap) {
    try {
        BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream(path));
        bitmap.compress(Bitmap.CompressFormat.JPEG, 80, bos);
        bos.flush();
        bos.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static Bitmap openImage(String path) {
    Bitmap bitmap = null;
    try {
        BufferedInputStream bis = new BufferedInputStream(new FileInputStream(path));
        bitmap = BitmapFactory.decodeStream(bis);
        bis.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
```

```
        return bitmap;
    }
```

接下来是演示时间，如图 4-13 所示，用户在注册页面录入注册信息，App 调用 getDrawingCache 方法把整个注册界面截图并保存到 SD 卡；然后在另一个页面的图片列表选择 SD 卡上的指定图片文件，页面就会展示上次保存的注册界面图片，如图 4-14 所示。



图 4-13　保存注册信息图片



图 4-14　读取注册信息图片

刚才从 SD 卡读取图片文件用到了 BitmapFactory 的 decodeStream 方法，其实 BitmapFactory 还提供了其他方法，用起来更简单、方便，说明如下：

- decodeFile：该方法直接传文件路径的字符串，即可将指定路径的图片读取到 Bitmap 对象。
- decodeResource：该方法可从资源文件中读取图片信息。第一个参数一般传 getResources()，第二个参数传 drawable 图片的资源 id，如 R.drawable.phone。

4.4　Application 基础

本节介绍 Android 重要组件 Application 的基本概念和常见用法。首先说明 Application 的生命周期，接着利用 Application 的持久特性实现 App 内部全局内存中的数据保存和获取。

4.4.1　Application 的生命周期

Application 是 Android 的一大组件，在 App 运行过程中有且仅有一个 Application 对象贯穿整个生命周期。打开 AndroidManifest.xml 时会发现 activity 节点的上级正是 application 节点，只是默认的 application 节点没有指定 name 属性，不像 activity 节点默认指定 name 属性值为.MainActivity，让人知晓这个 activity 的入口代码是 MainActivity.java。现在我们给 application 节点加上 name 属性，看看其庐山真面目。

（1）打开 AndroidManifest.xml，给 application 节点加上 name 属性，表示 application 的

入口代码是 MainApplication.java。

```
android:name=".MainApplication"
```

（2）创建 MainApplication 类，该类继承自 Application，可以重写的方法主要有以下 4 个。

- onCreate：在 App 启动时调用。
- onTerminate：在 App 退出时调用（按字面意思）。
- onLowMemory：在低内存时调用。
- onConfigurationChanged：在配置改变时调用，例如从竖屏变为横屏。

（3）运行 App，同时开启日志的打印。但是只在一开始看到 MainApplication 的 onCreate 操作（先于 Activity 的 onCreate），却始终无法看到它的 onTerminate 操作，无论是自行退出还是强行杀死 App 的进程，日志都不会打印 onTerminate。

信不信，无论你怎么折腾，这个 onTerminate 都不会出来。Android 明明提供了这个函数，同时提供了关于该函数的解释，说明文字如下：This method is for use in emulated process environments. It will never be called on a production Android device, where processes are removed by simply killing them; no user code (including this callback) is executed when doing so。这段话的意思是该方法是供模拟环境用的，在真机上永远不会被调用，无论是直接杀进程还是代码退出。

现在很明确了，onTerminate 方法就是个摆设，中看不中用。如果读者想在 App 退出前做资源回收操作，那么千万不要放在 onTerminate 方法中。

4.4.2 利用 Application 操作全局变量

C/C++有全局变量，因为全局变量保存在内存中，所以操作全局变量就是操作内存，内存的读写速度远比读写数据库或读写文件快得多。全局的意思是其他代码都可以引用该变量，因此全局变量是共享数据和消息传递的好帮手。不过，Java 没有全局变量的概念。与之比较接近的是类里面的静态成员变量，该变量可被外部直接引用，并且在不同地方引用的值是一样的（前提是在引用期间不能修改该变量的值），所以可以借助静态成员变量实现类似全局变量的功能。

前面花费很大功夫介绍 Application 的生命周期，目的是说明其生命周期覆盖 App 运行的全过程。不像短暂的 Activity 生命周期，只要进入别的页面，原页面就被停止或销毁。因此，通过利用 Application 的持久存在性可以在 Application 对象中保存全局变量。

适合在 Application 中保存的全局变量主要有下面 3 类数据：

（1）会频繁读取的信息，如用户名、手机号等。

（2）从网络上获取的临时数据，为节约流量、减少用户等待时间，想暂时放在内存中供下次使用，如 logo、商品图片等。

（3）容易因频繁分配内存而导致内存泄漏的对象，如 Handler 对象等。

要想通过 Application 实现全局内存的读写，得完成以下 3 项工作：

（1）写一个继承自 Application 的类 MainApplication。该类要采用单例模式，内部声明自身类的一个静态成员对象，在创建 App 时把自身赋值给这个静态对象，然后提供该静态对象

的获取方法 getInstance。

（2）在 Activity 中调用 MainApplication 的 getInstance 方法，获得 MainApplication 的一个静态对象，通过该对象访问 MainApplication 的公共变量和公共方法。

（3）不要忘了在 AndroidManifest.xml 中注册新定义的 Application 类名，即在 application 节点中增加 android:name 属性，值为.MainApplication。

下面继续演示全局内存的读写效果，如图 4-15 所示。App 把注册信息保存到 MainApplication 的全局变量中，然后在另一个页面从 MainApplication 的全局变量中读取保存好的注册信息，如图 4-16 所示。



图 4-15　注册信息保存到全局内存



图 4-16　从全局内存读取注册信息

下面是自定义 MainApplicaton 类的代码：

```
public class MainApplication extends Application {
    private static MainApplication mApp;
    public HashMap<String, String> mInfoMap = new HashMap<String, String>();

    public static MainApplication getInstance() {
        return mApp;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        mApp = this;
    }
}
```

完成以上编码后，Activity 页面代码即可直接通过 MainApplication.getInstance().mInfoMap 对全局变量进行增、删、改、查操作。

4.5 实战项目：购物车

购物车的应用面很广，凡是电商 App 都可以看到购物车的身影。本章以购物车为实战项目，除了购物车使用广泛的特点，还因为购物车用到多种存储方式。现在我们开启购物车的体验之旅吧!

4.5.1 设计思路

先来看常见的购物车的外观。第一次进入购物车频道，购物车里面是空的，如图 4-17 所示。接着去商品频道选购手机，随便挑几款加入购物车，然后返回购物车，即可看到购物车里的商品列表，有商品图片、名称、数量、单价、总价等信息，如图 4-18 所示。

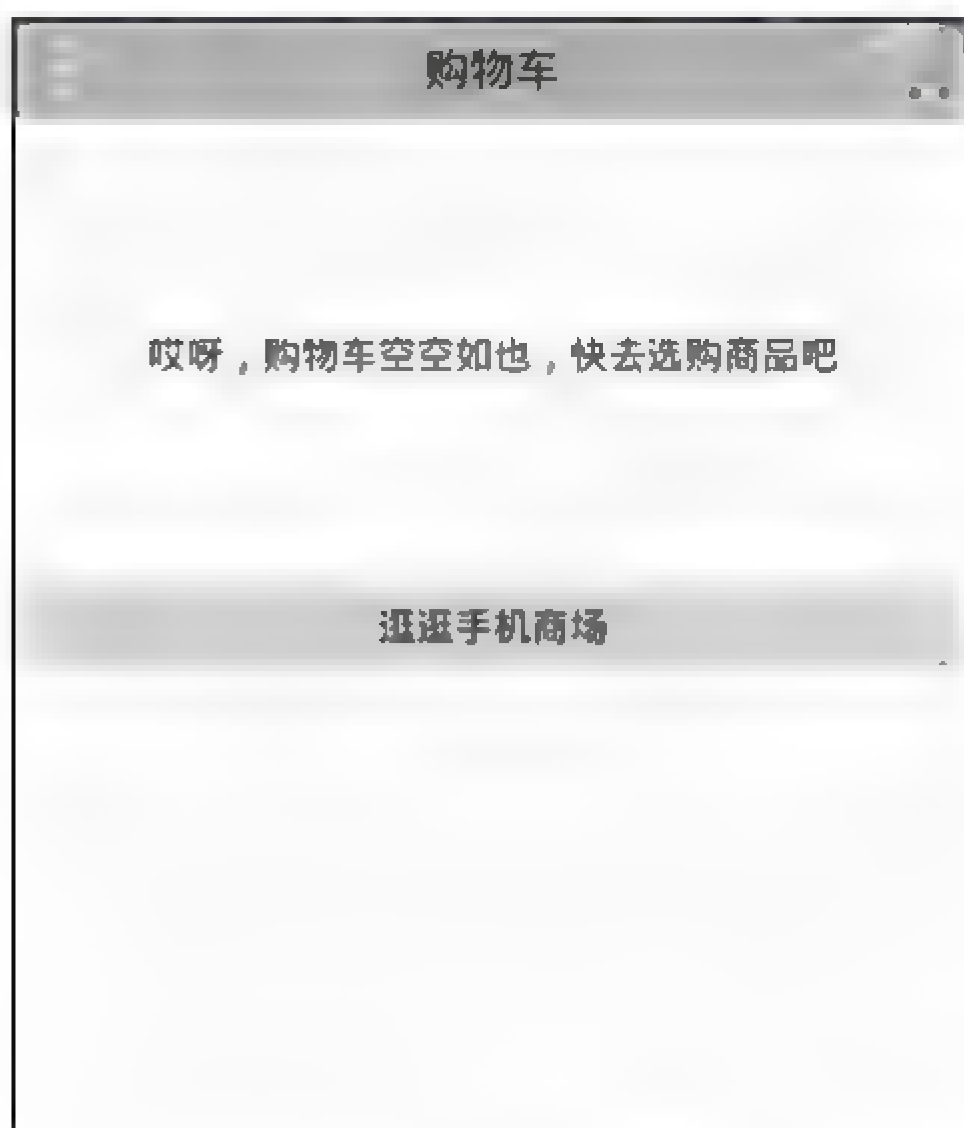


图 4-17 空空如也的购物车



图 4-18 购物车的商品列表

购物车的存在感很强，并不仅仅在购物车页面才能看到。往往在商场频道，甚至某个商品详情页面，都会看到某个角落冒出一个购物车图标。一旦有新商品加入购物车，购物车图标上的商品数量就立马加一。当然，用户也可以点击购物车图标直接跳转到购物车页面。商品频道除了商品列表外，页面右上角还有一个购物车图标，这个图标有时在页面右上角，有时又在页面右下角，如图 4-19 所示。商品详情页面通常也有购物车图标，如果用户在详情页面把商品加入购物车，那么图标上的数字也会加一，如图 4-20 所示。

现在我们来看购物车到底采取了哪些存储方式。

- 数据库 SQLite：最直观的是数据库，购物车里的商品列表一定放在 SQLite 中，增删改查都少不了 SQLite。



图 4-19　手机商场的商品列表



图 4-20　商品详情页面

- 共享参数 SharedPreferences：注意不同页面的右上角购物车图标都有数字，表示购物车中的商品数量，商品数量建议保存在共享参数中。因为每个页面都要显示商品数量，如果每次都到数据库中执行 count 操作，就会很消耗资源。因为商品数量需要持久地存储，所以不适合放在全局内存中，不然下次启动 App 时，内存中的变量又从 0 开始。
- SD 卡文件：通常情况下，商品图片来自于电商平台的服务器，这年头流量是很宝贵的，可是图片恰恰很耗流量（尤其是大图）。从用户的钱包着想，App 得把下载的图片保存在 SD 卡中。这样一来，下次用户访问商品详情页面时，App 便能直接从 SD 卡获取商品图片，不但不花流量而且加快浏览速度，一举两得。
- 全局内存：访问 SD 卡的图片文件固然是个好主意，然而商品频道、购物车频道等可能在一个页面展示多张商品小图，如果每张小图都要访问 SD 卡，频繁的 SD 卡读写操作也很耗资源。更好的办法是把商品小图加载进全局内存，这样直接从内存中获取图片，高效又快速。之所以不把商品大图放入全局内存，是因为大图很耗空间，一不小心就会占用几十兆内存。

不找不知道，一找吓一跳，原来购物车用到了这么多种存储方式。

4.5.2　小知识：菜单 Menu

之前的章节在进行某项控制操作时一般由按钮控件触发。如果页面上需要支持多个控制操作，比如去商场购物、清空购物车、查看商品详情、删除指定商品等，就得在页面上添加多个按钮。如此一来，App 页面显得杂乱无章，满屏按钮既碍眼又不便操作。这时，就可以使用菜单控件。

菜单无论在哪里都是常用控件，Android 的菜单主要分两种，一种是选项菜单 OptionMenu，通过按菜单键或点击事件触发，对应 Windows 上的开始菜单；另一种是上下文菜单 ContextMenu，通过长按事件触发，对应 Windows 上的右键菜单。无论是哪种菜单，都有对应的菜单布局文件，就像每个活动页面都有一个布局文件一样。不同的是页面的布局文件放在 res/layout 目录下，菜单的布局文件放在 res/menu 目录下。

下面来看 Android 的选项菜单和上下文菜单。

1. 选项菜单 OptionMenu

弹出选项菜单的途径有 3 种：

（1）按菜单键。
（2）在代码中手动打开选项菜单，即调用 openOptionsMenu 方法。
（3）按工具栏右侧的溢出菜单按钮，这个在后续介绍工具栏时进行介绍。

实现选项菜单的功能需要重写以下两种方法。

- onCreateOptionsMenu：在页面打开时调用。需要指定菜单列表的 XML 文件。
- onOptionsItemSelected：在列表的菜单项被选中时调用。需要对不同的菜单项做分支处理。

下面是菜单布局文件的代码，很简单，就是 menu 与 item 的组合排列：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/menu_change_time"
        android:orderInCategory="1"
        android:title="改变时间"/>
    <item
        android:id="@+id/menu_change_color"
        android:orderInCategory="8"
        android:title="改变颜色"/>
    <item
        android:id="@+id/menu_change_bg"
        android:orderInCategory="9"
        android:title="改变背景"/>
</menu>
```

接下来是使用选项菜单的代码片段：

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_option, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.menu_change_time) {
            setRandomTime();
        } else if (id == R.id.menu_change_color) {
            tv_option.setTextColor(getRandomColor());
```

```
        } else if (id == R.id.menu_change_bg) {
            tv_option.setBackgroundColor(getRandomColor());
        }
        return true;
    }

    private void setRandomTime() {
        String desc = DateUtil.getNowDateTime("yyyy-MM-dd HH:mm:ss") + " 这里是菜单显示文本";
        tv_option.setText(desc);
    }

    private int[] mColorArray = {
            Color.BLACK, Color.WHITE, Color.RED, Color.YELLOW, Color.GREEN,
            Color.BLUE, Color.CYAN, Color.MAGENTA, Color.GRAY, Color.DKGRAY };
    private int getRandomColor() {
        int random = (int) (Math.random()*10 % 10);
        return mColorArray[random];
    }
```

按菜单键和调用 openOptionsMenu 方法弹出的选项菜单都是在页面下方，如图 4-21 所示。

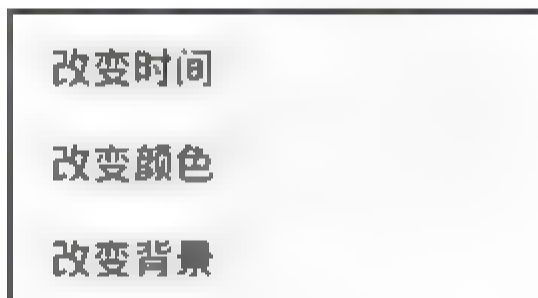


图 4-21　选项菜单的菜单列表

2. 上下文菜单 ContextMenu

弹出上下文菜单的途径有两种：

（1）默认在某个控件被长按时弹出。通常在 onStart 函数中加入 registerForContextMenu 方法为指定控件注册上下文菜单，在 onStop 函数中加入 unregisterForContextMenu 方法为指定控件注销上下文菜单。

（2）在除长按事件之外的其他事件中打开上下文菜单。先执行 registerForContextMenu 方法注册菜单，然后执行 openContextMenu 方法打开菜单，最后执行 unregisterForContextMenu 方法注销菜单。

实现上下文菜单的功能需要重写以下两种方法。

- onCreateContextMenu：在此指定菜单列表的 XML 文件，作为上下文菜单列表项的来源。
- onContextItemSelected：在此对不同的菜单项做分支处理。

上下文菜单的布局文件格式同选项菜单，下面是使用上下文菜单的代码片段：

```
@Override
protected void onResume() {
    registerForContextMenu(tv_context);
    super.onResume();
}

@Override
protected void onPause() {
    unregisterForContextMenu(tv_context);
    super.onPause();
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    getMenuInflater().inflate(R.menu.menu_option, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.menu_change_time) {
        setRandomTime();
    } else if (id == R.id.menu_change_color) {
        tv_context.setTextColor(getRandomColor());
    } else if (id == R.id.menu_change_bg) {
        tv_context.setBackgroundColor(getRandomColor());
    }
    return true;
}
```

上下文菜单的菜单列表固定显示在页面中部，菜单外的其他页面区域颜色会变深，具体效果如图 4-22 所示。

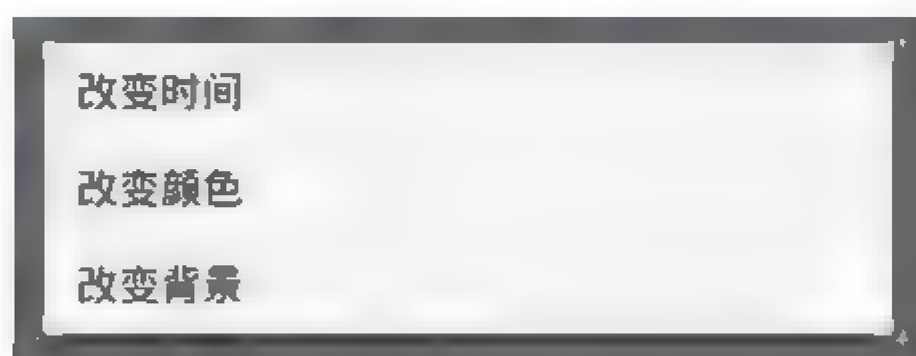


图 4-22　上下文菜单的菜单列表

4.5.3　代码示例

这一章的编码开始有些复杂了，不但有各种控件和布局的操作，还有 4 种存储方式的使用，再加上 Activity 与 Application 两大组件的运用，已然是一个正规 App 的雏形。



编码过程分为 4 步（增加的一步是对 AndroidManifest.xml 认真配置）：

01 想好代码文件与布局文件的名称，比如购物车页面的代码文件取名 ShoppingCartActivity.java，对应的布局文件名是 activity_shopping_cart.xml；商场频道页面的代码文件取名 ShoppingChannelActivity.java，对应的布局文件名是 activity_shopping_channel.xml；商品详情页面的代码文件取名 ShoppingDetailActivity，对应的布局文件名是 activity_shopping_detail.xml；另有一个全局应用的代码文件 MainApplication.java。

02 在 AndroidManifest.xml 中补充相应配置，主要有以下 3 点：

（1）注册 3 个页面的 acitivity 节点，注册代码如下：

```
<activity android:name=".ShoppingCartActivity" android:theme="@style/AppBaseTheme" />
<activity android:name=".ShoppingChannelActivity" />
<activity android:name=".ShoppingDetailActivity" />
```

（2）给 application 补充 name 属性，值为 MainApplication，举例如下：

```
android:name=".MainApplication"
```

（3）声明 SD 卡的操作权限，主要补充下面 3 行权限配置：

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAG" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

03 res 目录下的 XML 文件编写也多了起来，主要工作包括：

（1）在 res/layout 目录下创建布局文件 activity_shopping_cart.xml、activity_shopping_channel.xml、activity_shopping_detail.xml，分别根据页面效果图编写 3 个页面的布局定义文件。

（2）在 res/menu 目录下创建菜单布局文件 menu_cart.xml 和 menu_goods.xml，分别用于购物车的选项菜单和商品项的上下文菜单。

（3）在 values/styles.xml 中补充下面的样式定义，给不带导航栏的购物车页面使用：

```
<style name="AppBaseTheme" parent="Theme.AppCompat.Light" />
```

04 在项目的包名目录下创建类 MainApplication、ShoppingCartActivity、ShoppingChannel-Activity 和 ShoppingDetailActivity，并填入具体的控件操作与业务逻辑代码。

下面是购物车页面 ShoppingCartActivity.java 的主要代码片段：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(R.layout.activity_shopping_cart);
    iv_menu = (ImageView) findViewById(R.id.iv_menu);
    tv_title = (TextView) findViewById(R.id.tv_title);
    tv_count = (TextView) findViewById(R.id.tv_count);
```

```
        tv_total_price = (TextView) findViewById(R.id.tv_total_price);
        ll_content = (LinearLayout) findViewById(R.id.ll_content);
        ll_cart = (LinearLayout) findViewById(R.id.ll_cart);
        ll_empty = (LinearLayout) findViewById(R.id.ll_empty);
        iv_menu.setOnClickListener(this);
        findViewById(R.id.btn_shopping_channel).setOnClickListener(this);
        findViewById(R.id.btn_settle).setOnClickListener(this);
        iv_menu.setVisibility(View.VISIBLE);
        tv_title.setText("购物车");
        mCount = Integer.parseInt(SharedUtil.getIntance(this).readShared("count", "0"));
        showCount(mCount);
    }

    //显示购物车图标中的商品数量
    private void showCount(int count) {
        mCount = count;
        tv_count.setText(""+mCount);
        if (mCount == 0) {
            ll_content.setVisibility(View.GONE);
            ll_cart.removeAllViews();
            ll_empty.setVisibility(View.VISIBLE);
        } else {
            ll_content.setVisibility(View.VISIBLE);
            ll_empty.setVisibility(View.GONE);
        }
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.iv_menu) {
            openOptionsMenu();
        } else if (v.getId() == R.id.btn_shopping_channel) {
            Intent intent = new Intent(this, ShoppingChannelActivity.class);
            startActivity(intent);
        } else if (v.getId() == R.id.btn_settle) {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("结算商品");
            builder.setMessage("客官抱歉，支付功能尚未开通，请下次再来");
            builder.setPositiveButton("我知道了", null);
            builder.create().show();
        }
    }
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_cart, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.menu_shopping) {
        Intent intent = new Intent(this, ShoppingChannelActivity.class);
        startActivity(intent);
    } else if (id == R.id.menu_clear) {
        //清空购物车数据库
        mCartHelper.deleteAll();
        ll_cart.removeAllViews();
        SharedUtil.getIntance(this).writeShared("count", "0");
        showCount(0);
        mGoodsView.clear();
        mGoodsMap.clear();
        Toast.makeText(this, "购物车已清空", Toast.LENGTH_SHORT).show();
    } else if (id == R.id.menu_return) {
        finish();
    }
    return true;
}

private HashMap<Integer, Long> mGoodsView = new HashMap<Integer, Long>();
private View mContextView;
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo) {
    mContextView = v;
    getMenuInflater().inflate(R.menu.menu_goods, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == R.id.menu_detail) {
        //跳转到查看商品详情页面
        goDetail(mGoodsView.get(mContextView.getId()));
    } else if (id == R.id.menu_delete) {
        //从购物车删除商品的数据库操作
```



```
                long goods_id = mGoodsView.get(mContextView.getId());
                mCartHelper.delete("goods_id="+goods_id);
                // cart.removeView(mContextView);
                //更新购物车中的商品数量
                int left_count = mCount-1;
                for (int i=0; i<mCartArray.size(); i++) {
                    if (goods_id == mCartArray.get(i).goods_id) {
                        left_count = mCount-mCartArray.get(i).count;
                        mCartArray.remove(i);
                        break;
                    }
                }
                SharedUtil.getIntance(this).writeShared("count", ""+left_count);
                showCount(left_count);
                Toast.makeText(this, "已从购物车删除"+mGoodsMap.get(goods_id).name,
Toast.LENGTH_SHORT).show();
                mGoodsMap.remove(goods_id);
                refreshTotalPrice();
            }
            return true;
        }

        private void goDetail(long rowid) {
            Intent intent = new Intent(this, ShoppingDetailActivity.class);
            intent.putExtra("goods_id", rowid);
            startActivity(intent);
        }

        private GoodsDBHelper mGoodsHelper;
        private CartDBHelper mCartHelper;
        private String mFirst = "true";
        @Override
        protected void onResume() {
            super.onResume();
            mGoodsHelper = GoodsDBHelper.getInstance(this, 1);
            mGoodsHelper.openWriteLink();
            mCartHelper = CartDBHelper.getInstance(this, 1);
            mCartHelper.openWriteLink();
            mFirst = SharedUtil.getIntance(this).readShared("first", "true");
            downloadGoods();
            SharedUtil.getIntance(this).writeShared("first", "false");
            showCart();
        }
```

```
    @Override
    protected void onPause() {
        super.onPause();
        mGoodsHelper.closeLink();
        mCartHelper.closeLink();
    }

    //模拟网络数据，初始化数据库中的商品信息
    private void downloadGoods() {
        String path = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_
DOWNLOADS) + "/";
        if (mFirst.equals("true")) {
            for (int i=0; i<mNameArray.length; i++) {
                GoodsInfo info = new GoodsInfo();
                info.name = mNameArray[i];
                info.desc = mDescArray[i];
                info.price = mPriceArray[i];
                long rowid = mGoodsHelper.insert(info);
                info.rowid = rowid;
                //往全局内存写入商品小图
                Bitmap thumb = BitmapFactory.decodeResource(getResources(), mThumbArray[i]);
                MainApplication.getInstance().mIconMap.put(rowid, thumb);
                String thumb_path = path + rowid + "_s.jpg";
                FileUtil.saveImage(thumb_path, thumb);
                info.thumb_path = thumb_path;
                //往 SD 卡保存商品大图
                Bitmap pic = BitmapFactory.decodeResource(getResources(), mPicArray[i]);
                String pic_path = path + rowid + ".jpg";
                FileUtil.saveImage(pic_path, pic);
                pic.recycle();
                info.pic_path = pic_path;
                mGoodsHelper.update(info);
            }
        } else {
            ArrayList<GoodsInfo> goodsArray = mGoodsHelper.query("1=1");
            for (int i=0; i<goodsArray.size(); i++) {
                GoodsInfo info = goodsArray.get(i);
                Bitmap thumb = BitmapFactory.decodeFile(info.thumb_path);
                MainApplication.getInstance().mIconMap.put(info.rowid, thumb);
            }
        }
    }
```

4.6 小　结

本章主要介绍了Android常用的几种数据存储方式，包括共享参数SharedPreferences的键值对存取、数据库SQLite的关系型数据存取、SD卡的文件写入与读取操作（含文本文件读写和图片文件读写）、App全局内存的读写以及为实现全局内存而学习的Application组件的生命周期及其用法。最后设计了一个实战项目“购物车”，通过该项目的编码进一步复习巩固本章4种存储方式的使用，另外介绍了选项菜单和上下文菜单的基本用法。

通过本章的学习，读者应该能够掌握以下3种开发技能：

（1）学会共享参数SharedPreferences、数据库SQLite、SD卡文件、全局内存4种存储方式的用法。

（2）学会Application组件的用法。

（3）学会选项菜单和上下文菜单的基本用法。



高级控件

本章介绍 App 开发常用的一些高级控件及相关工具，主要包括日期时间控件的用法、列表类视图及其适配器的用法、翻页类视图及其适配器的用法、碎片及其适配器的用法等，另外介绍四大组件之一 Broadcast 的基本概念与常见用法。最后结合本章所学的知识演示一个实战项目“日历/日程表”的设计与实现。

5.1 日期时间控件

本节介绍 Android 的日期时间控件，主要是日期选择对话框 DatePickerDialog 和时间选择对话框 TimePickerDialog 的用法。

5.1.1 日期选择器 DatePicker

虽然 EditText 控件提供 inputType="date"的日期输入，但是很少有用户会老老实实地手工输入日期，况且 EditText 还不支持“****年**月**日”这样的日期格式，所以都要系统提供日期控件，供用户选择具体的年月日，在 Android 中这个控件是 DatePicker。不过，DatePicker 并非弹窗模式，而是直接在页面上占据一块区域，并且不会自动关闭。按习惯来说，日期控件应该在当前页面弹出，选择完日期就要把控件关掉。因此，DatePicker 不适合直接使用，实际开发中用的是已经封装好的日期选择对话框 DatePickerDialog。

DatePickerDialog 相当于在 AlertDialog 上加载了 DatePicker，用起来更简单，只需调用构造函数设置一下当前年、月、日，然后调用 show 方法即可弹出日期对话框。日期选择事件由监听器 OnDateSetListener 负责响应，在该监听器实现的 onDateSet 方法中，开发者能够获得用户选择的具体日期，并做后续处理。这里要特别注意 onDateSet 方法的月份参数，该参数的起始值不是 1 而是 0。也就是说，一月份对应的参数数值是 0，十二月份对应的参数数值是 11。如果实在不理解，记住这里的月份值要加 1 就行了。



图 5-1 日期选择对话框

图 5-1 所示为一个默认样式的日期选择对话框。其中，年、月、日通过上下滑动选择。

下面是使用日期对话框的代码：

```
public class DatePickerActivity extends AppCompatActivity implements OnClickListener,
OnDateSetListener {
    private TextView tv_date;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_date_picker);
        tv_date = (TextView) findViewById(R.id.tv_date);
        findViewById(R.id.btn_date).setOnClickListener(this);
    }

    @Override
```

```
    public void onClick(View v) {
        if (v.getId() == R.id.btn_date) {
            Calendar calendar = Calendar.getInstance();
            DatePickerDialog dialog = new DatePickerDialog(this, this,
                    calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH),
                    calendar.get(Calendar.DAY_OF_MONTH));
            dialog.show();
        }
    }

    @Override
    public void onDateSet(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
        String desc = String.format("您选择的日期是%d 年%d 月%d 日",
                year, monthOfYear+1, dayOfMonth);
        tv_date.setText(desc);
    }
}
```

5.1.2 时间选择器 TimePicker

有了日期选择器，肯定有对应的时间选择器。同样，实际开发中也不直接用 TimePicker，而是用封装好的时间选择对话框 TimePickerDialog。该对话框的用法类似 DatePickerDialog，不同之处主要有两个：

（1）构造函数传的是当前的小时与分钟，最后一个参数表示是否采用二十四小时制，一般传 true，表示小时的数值范围为 0～23。

（2）时间选择监听器是 OnTimeSetListener，对应需要实现的方法是 onTimeSet，在该方法中可获得用户选好的小时和分钟。

图 5-2 所示为一个默认样式的时间选择对话框。其中，小时与分钟可通过上下滑动选择。



图 5-2　时间选择对话框

下面是使用时间对话框的代码：

```
public class TimePickerActivity extends AppCompatActivity implements OnClickListener,
OnTimeSetListener {
    private TextView tv_time;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_time_picker);
        tv_time = (TextView) findViewById(R.id.tv_time);
```

```
            findViewById(R.id.btn_time).setOnClickListener(this);
        }

        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.btn_time) {
                Calendar calendar = Calendar.getInstance();
                TimePickerDialog dialog = new TimePickerDialog(this, this,
                        calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE),
true);
                dialog.show();
            }
        }

        @Override
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            String desc = String.format("您选择的时间是%d 时%d 分", hourOfDay, minute);
            tv_time.setText(desc);
        }
    }
```

5.2 列表类视图

本节介绍列表类视图怎样结合基本适配器实现视图展示的效果，包括基本适配器 BaseAdapter 的用法、列表视图 ListView 的分隔线设置与使用注意点、网格视图 GridView 的分隔线设置与使用注意点。

5.2.1 基本适配器 BaseAdapter

第 3 章介绍下拉框 Spinner 时提到该控件可使用 ArrayAdapter 和 SimpleAdapter 两种适配器。其中，ArrayAdapter 适用于纯文本的列表数据，SimpleAdapter 适用于带图标的列表数据。实际应用中常常有更复杂的列表，比如同一项中存在多个控件，这种情况即使用 SimpleAdapter 也很吃力，而且不易扩展。基于此，Android 提供了一种适应性更强的基本适配器 BaseAdapter，该适配器允许开发者在别的代码文件中进行逻辑处理，大大提高了代码的可读性和可维护性。

从 BaseAdapter 派生的数据适配器主要实现下面 3 个方法。

- 构造函数：指定适配器需要处理的数据集合。
- getCount：获取数据项的个数。
- getView：获取每项的展示视图，并对每项的内部控件进行业务处理。

下面以 Spinner 控件为载体，演示如何操作 BaseAdapter，具体的编码分为 3 步：



步骤 01 编写列表项的布局文件，示例代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ll_item"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <ImageView
        android:id="@+id/iv_icon"
        android:layout_width="0dp"
        android:layout_height="80dp"
        android:layout_weight="1"
        android:scaleType="fitCenter" />

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3"
        android:orientation="vertical" >

        <TextView
            android:id="@+id/tv_name"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:gravity="left|center"
            android:textColor="@color/black"
            android:textSize="20sp" />

        <TextView
            android:id="@+id/tv_desc"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="2"
            android:gravity="left|center"
            android:textColor="@color/black"
            android:textSize="13sp" />
    </LinearLayout>
</LinearLayout>
```

步骤 02 写个新的适配器继承 BaseAdapter，实现对列表项视图的获取与操作，示例代码如下：

```
public class PlanetAdapter extends BaseAdapter {
    private LayoutInflater mInflater;
    private Context mContext;
    private int mLayoutId;
    private ArrayList<Planet> mPlanetList;
    private int mBackground;

    public PlanetAdapter(Context context, int layout_id, ArrayList<Planet> planet_list, int background) {
        mInflater = LayoutInflater.from(context);
        mContext = context;
        mLayoutId = layout_id;
        mPlanetList = planet_list;
        mBackground = background;
    }

    @Override
    public int getCount() {
        return mPlanetList.size();
    }

    @Override
    public Object getItem(int arg0) {
        return mPlanetList.get(arg0);
    }

    @Override
    public long getItemId(int arg0) {
        return arg0;
    }

    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {
        ViewHolder holder = null;
        if (convertView == null) {
            holder = new ViewHolder();
            convertView = mInflater.inflate(mLayoutId, null);
            holder.ll_item = (LinearLayout) convertView.findViewById(R.id.ll_item);
            holder.iv_icon = (ImageView) convertView.findViewById(R.id.iv_icon);
            holder.tv_name = (TextView) convertView.findViewById(R.id.tv_name);
            holder.tv_desc = (TextView) convertView.findViewById(R.id.tv_desc);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolder) convertView.getTag();
```

```
            }
            Planet planet = mPlanetList.get(position);
            holder.ll_item.setBackgroundColor(mBackground);
            holder.iv_icon.setImageResource(planet.image);
            holder.tv_name.setText(planet.name);
            holder.tv_desc.setText(planet.desc);
            return convertView;
        }

        public final class ViewHolder {
            private LinearLayout ll_item;
            public ImageView iv_icon;
            public TextView tv_name;
            public TextView tv_desc;
        }
    }
```

03 在页面代码中构造该适配器，并应用于 Spinner 对象，示例代码如下：

```
        private void initSpinner() {
            planetList = Planet.getDefaultList();
            PlanetAdapter adapter = new PlanetAdapter(this, R.layout.item_list, planetList, Color.WHITE);
            Spinner sp = (Spinner) findViewById(R.id.sp_planet);
            sp.setPrompt("请选择行星");
            sp.setAdapter(adapter);
            sp.setSelection(0);
            sp.setOnItemSelectedListener(new MySelectedListener());
        }

        private class MySelectedListener implements OnItemSelectedListener {
            public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
                Toast.makeText(BaseAdapterActivity.this, "您选择的是"+planetList.get(arg2).name,
Toast.LENGTH_LONG).show();
            }

            public void onNothingSelected(AdapterView<?> arg0) {
            }
        }
```

具体的列表对话框效果如图 5-3 所示。可以看到，每行左边是行星图标，右边的上面是行星名称，下面是行星的描述。因为对列表项布局 item_list.xml 使用了单独的适配器代码 PlanetAdapter，所以再多加几个控件也不怕麻烦了。

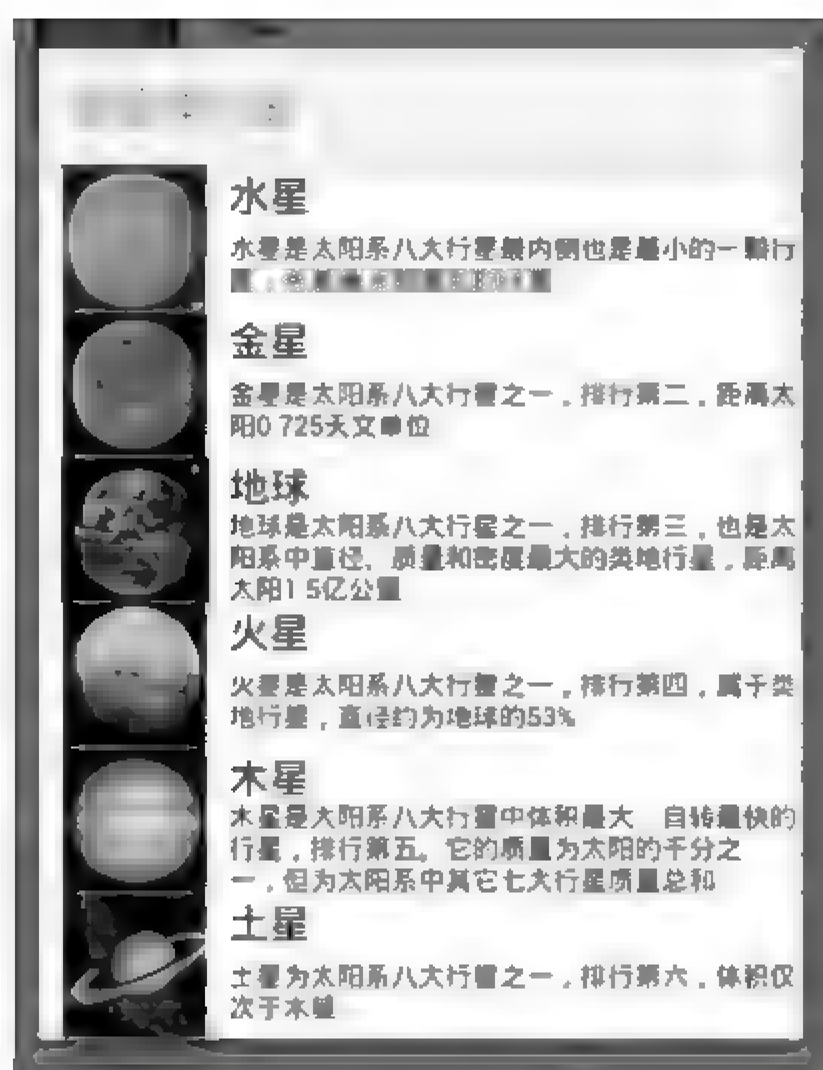


图 5-3　下拉列表中的基本适配器效果

5.2.2　列表视图 ListView

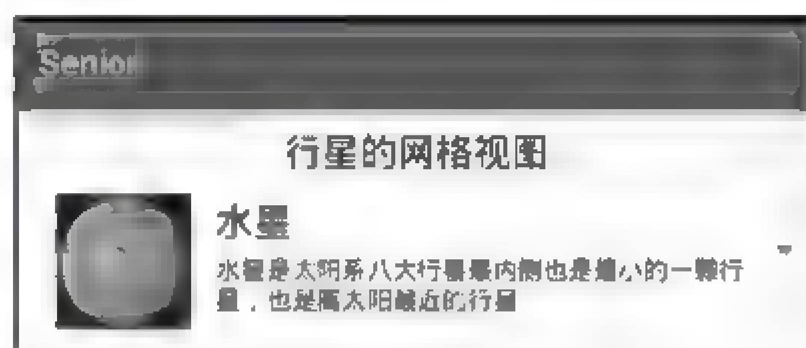


图 5-4　下拉框在页面上只显示一行

前面我们给 Spinner 控件加上了基本适配器，然而列表效果只在弹出对话框中展示，一旦选中某项，回到页面时又只显示选中的内容，如图 5-4 所示。

这么丰富的列表信息没展示在页面上实在是可惜，也许用户对好几项内容都感兴趣。如果想在页面上直接显示全部列表信息，就要引入新的列表视图 ListView。ListView 允许在页面上分行展示相似的数据界面，如新闻列表、商品列表、书籍列表等，方便用户逐行浏览与操作。列表视图 ListView 新增的属性与方法说明见表 5-1。

表 5-1　ListView 的属性与方法说明

XML 中的属性	ListView 类的设置方法	说明
divider	setDivider	指定分隔线的图形。如需取消分隔线，可设置该属性值为@null
dividerHeight	setDividerHeight	指定分隔线的高度
headerDividersEnabled	setHeaderDividersEnabled	指定是否显示列表开头的分隔线
footerDividersEnabled	setFooterDividersEnabled	指定是否显示列表末尾的分隔线

另外，ListView 实现了 3 个与适配器相关的方法。

- setAdapter：设置列表项的数据适配器，适配器一般继承 BaseAdapter。
- setOnItemClickListener：设置列表项的点击事件监听器 OnItemClickListener。
- setOnItemLongClickListener：设置列表项的长按事件监听器 OnItemLongClickListener。

下面是列表项处理点击事件和长按事件的代码：

```
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String desc = String.format("您点击了第%d 个行星，它的名字是%s", position + 1,
                mPlanetList.get(position).name);
        Toast.makeText(mContext, desc, Toast.LENGTH_LONG).show();
    }

    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id) {
        String desc = String.format("您长按了第%d 个行星，它的名字是%s", position + 1,
                mPlanetList.get(position).name);
        Toast.makeText(mContext, desc, Toast.LENGTH_LONG).show();
        return true;
    }
```

光看这些文字会觉得 ListView 是个加强版的 Spinner，不但可以直接在页面上展示列表，而且能设置分隔线与点击监听器。事实上，ListView 很令人头痛，使用过程中经常出现意想不到的状况，比如分隔线就容易出状况，下面演示分隔线的测试代码片段：

```
    private class DividerSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
                    LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
            lv_planet.setDivider(drawable);
            lv_planet.setDividerHeight(dividerHeight);
            lv_planet.setPadding(0, 0, 0, 0);
            lv_planet.setBackgroundColor(Color.TRANSPARENT);
            if (arg2 == 0) {  // 不显示分隔线(分隔线高度为 0)
                lv_planet.setDividerHeight(0);
            } else if (arg2 == 1) {  // 不显示分隔线(分隔线为 null)
                lv_planet.setDivider(null);
                lv_planet.setDividerHeight(dividerHeight);
            } else if (arg2 == 2) {  // 只显示内部分隔线(先设置分隔线高度)
                lv_planet.setDividerHeight(dividerHeight);
                lv_planet.setDivider(drawable);
            } else if (arg2 == 3) {  // 只显示内部分隔线(后设置分隔线高度)
                lv_planet.setDivider(drawable);
                lv_planet.setDividerHeight(dividerHeight);
            } else if (arg2 == 4) {  // 显示底部分隔线(高度是 wrap_content)
                lv_planet.setFooterDividersEnabled(true);
            } else if (arg2 == 5) {  // 显示底部分隔线(高度是 match_parent)
                params = new LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT, 0, 1);
                lv_planet.setFooterDividersEnabled(true);
            } else if (arg2 == 6) {  // 显示顶部分隔线(别瞎折腾了，显示不了)
```

```
                params = new LinearLayout.LayoutParams(LayoutParams.MATCH_PARENT, 0, 1);
                lv_planet.setFooterDividersEnabled(true);
                lv_planet.setHeaderDividersEnabled(true);
            } else if (arg2 == 7) {   // 显示全部分隔线(看我用 padding 大法)
                lv_planet.setPadding(0, dividerHeight, 0, dividerHeight);
                lv_planet.setBackgroundDrawable(drawable);
            }
            lv_planet.setLayoutParams(params);
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }
```

根据分隔线测试代码的演示结果，笔者总结了一下，大概有以下 5 种情况：

（1）代码中的 setDivider 方法只能设置具体的图片，不能设置颜色，即使把颜色值转为 ColorDrawable 也不行。在布局文件中可对 divider 属性直接指定颜色值。

（2）divider 属性设置为@null 时不能再设置 dividerHeight 属性为大于 0 的数值，因为这样一来最后一项就不会完全显示，底部有一部分被掩盖了。原因是列表高度为 wrap_content 时，系统已按照没有分隔线的情况计算列表高度，此时 dividerHeight 占用了 n-1 块空白分隔区域，最后一项被挤到背影里面去了，具体效果如图 5-5 所示。

（3）代码中要设置分隔线，务必先调用 setDivider 方法再调用 setDividerHeight 方法。如果先调用 setDividerHeight 再调用 setDivider，分隔线高度就会变成分隔图片的高度，而不是 setDividerHeight 设置的高度，具体效果如图 5-6 所示。布局文件不存在先后顺序问题。

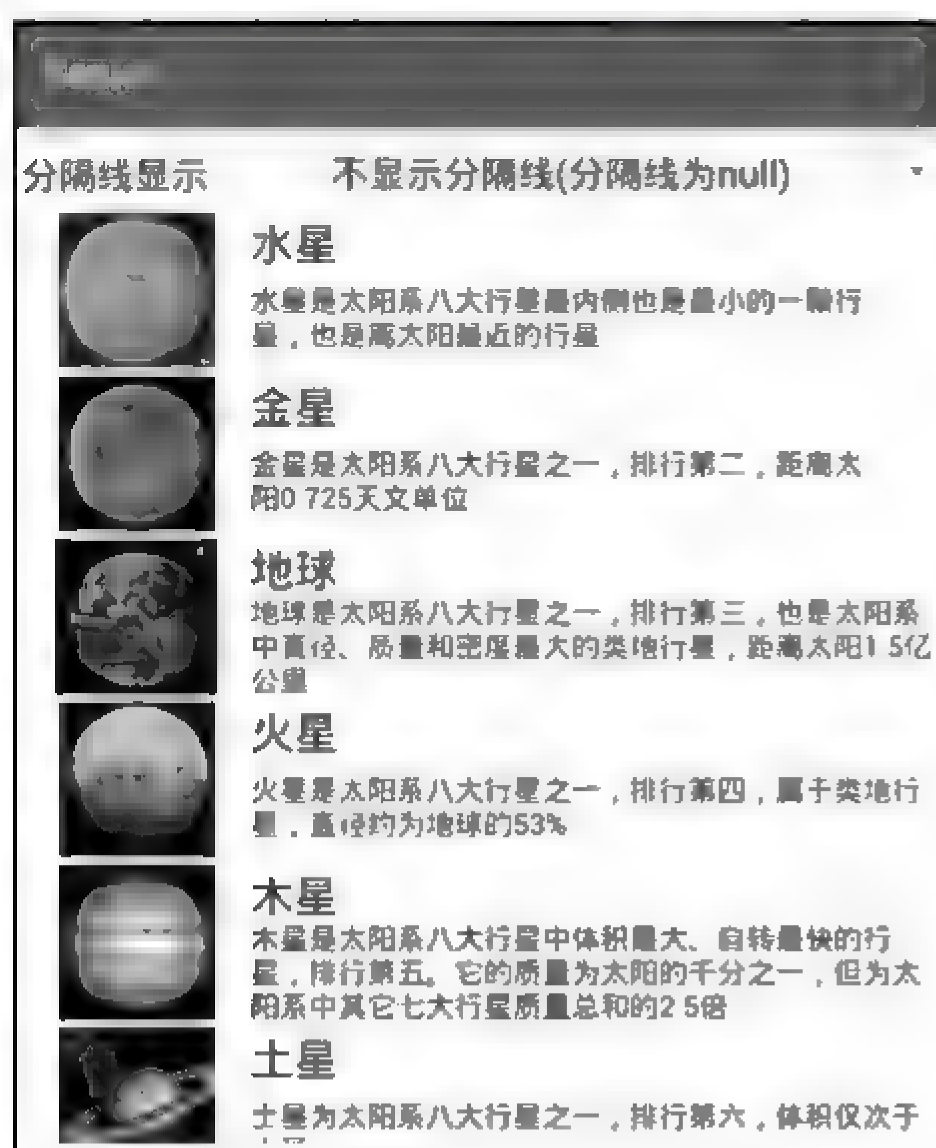


图 5-5　divider 属性设置为@null

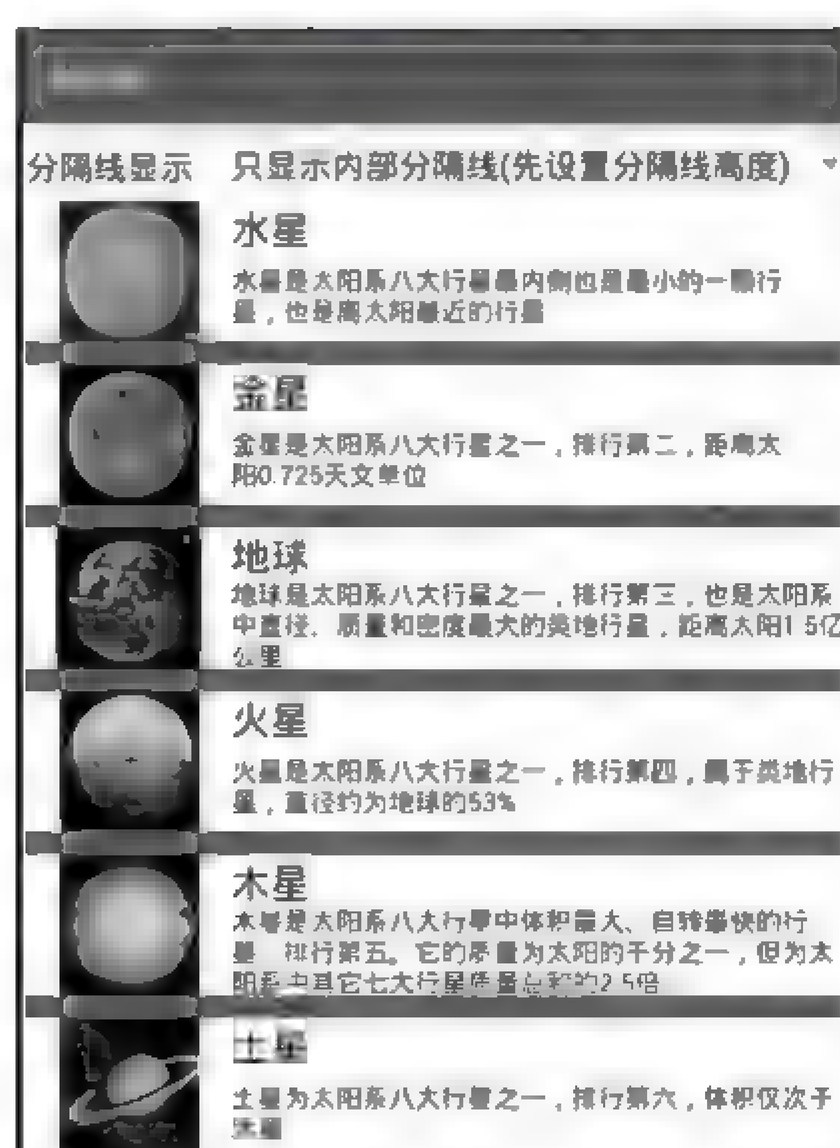


图 5-6　先调用 setDividerHeight

（4）显示列表底部的分隔线是有条件的，即当前 ListView 的高度不能为 wrap_content，否则就算把 footerDividersEnabled 设置为 true、调用 setFooterDividersEnabled 方法设置为 true，这条底部的分隔线也不会出现。除非把列表的高度设置为 match_parent 或设置足够高，才会显示底部的分隔线，调整列表高度后的具体效果如图 5-7 所示。

（5）列表顶部的分隔线就更难办了，ListView 不会显示顶部的分隔线。无论是 headerDividersEnabled 属性还是 setHeaderDividersEnabled 方法都没有作用，而且调整列表高度也没什么用，非常难以解决。

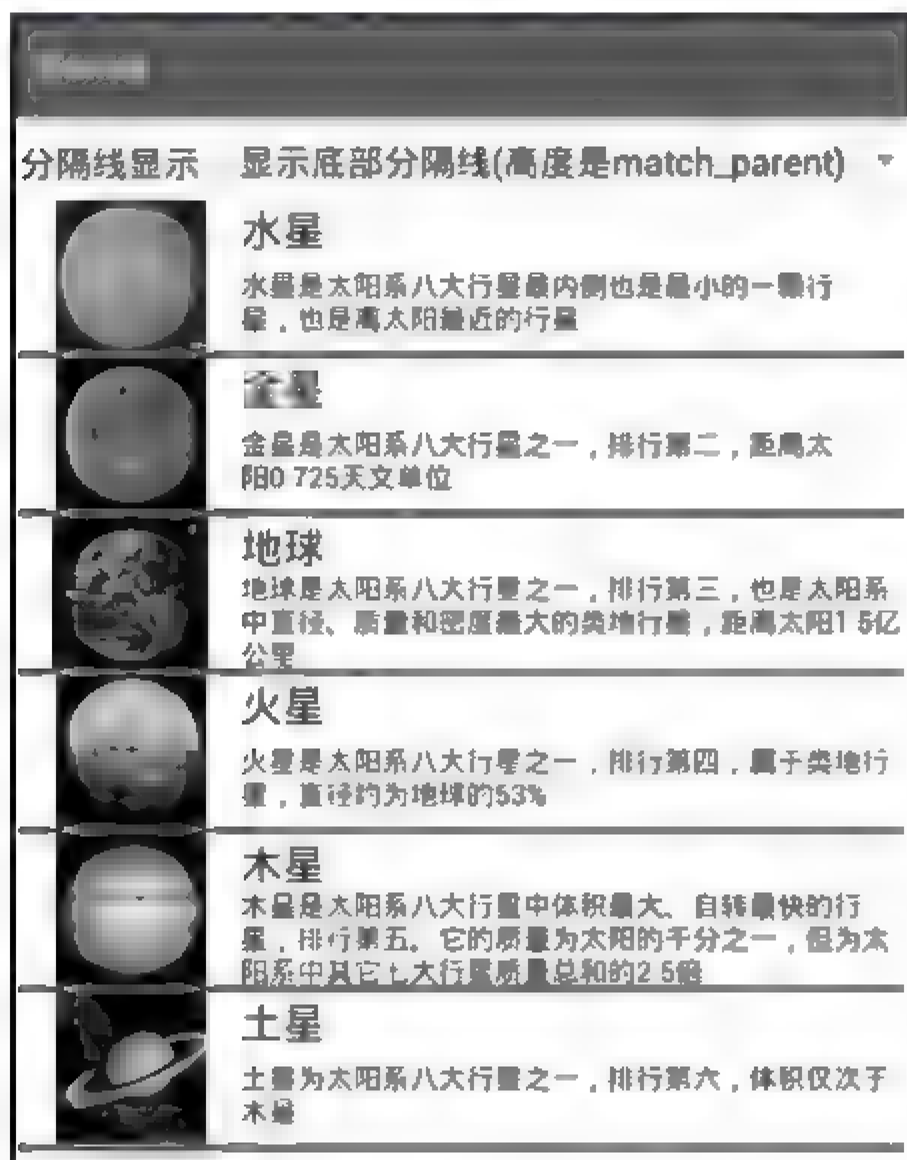


图 5-7 高度设置为 match_parent

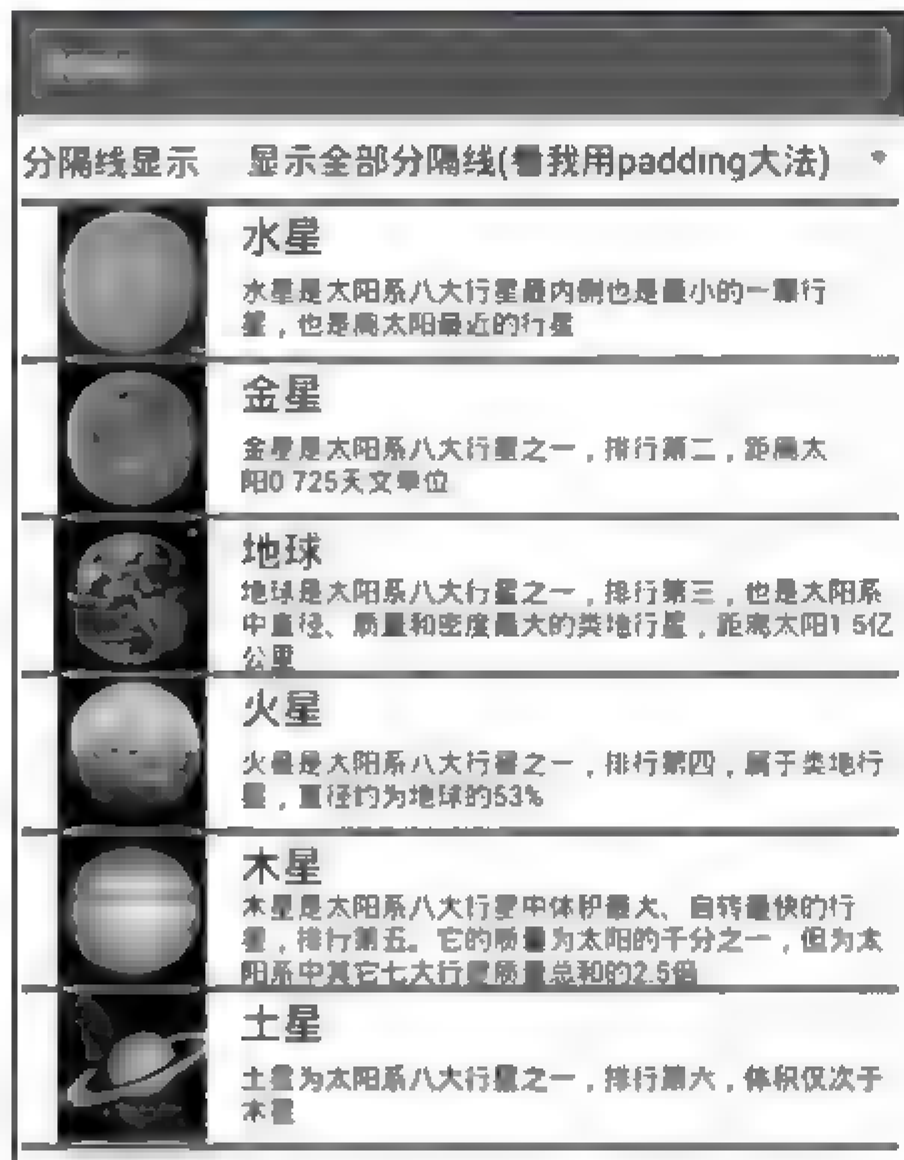


图 5-8 padding 显示头尾分隔线

既然底部和顶部的分隔线令人这般头痛，不如直接扔掉，另外想想别的办法。使用 padding 即可解决这个问题。首先给 ListView 设置背景图片，然后分别设置 paddingTop 与 paddingBottom，接下来顶部和底部就会出现两个背景图的 padding，具体效果如图 5-8 所示。

上面第 3 点和第 5 点已经明确是 Android 的 bug，较真的读者不必把时间浪费在上面。这不是设置问题，也不是方法调用问题，而且 SDK 的代码逻辑问题，详述如下：

（1）关于 setDivider 方法与 setDividerHeight 方法的先后顺序关系，参见下面的 setDivider 方法源码，问题在于 if 条件，这里“divider != null”的条件不准确，应当改为“divider!=null && mDividerHeight<=0”，如果已经指定分隔线的高度，就不用使用分隔图片的高度了。

```
public void setDivider(@Nullable Drawable divider) {
    if (divider != null) {
        mDividerHeight = divider.getIntrinsicHeight();
    } else {
        mDividerHeight = 0;
    }
    mDivider = divider;
```

```
            mDividerIsOpaque = divider == null || divider.getOpacity() == PixelFormat.OPAQUE;
            requestLayout();
            invalidate();
        }
```

（2）关于无法显示顶部的分隔线问题，可查看 ListView 源码的 dispatchDraw 方法，这里把问题代码贴出来了，具体如下：

```
            bounds.top = bottom;
            bounds.bottom = bottom + dividerHeight;
            drawDivider(canvas, bounds, i);
```

可以看到分隔线固定在该项底部，如果在顶部看到分隔线，那才是怪事。正确的写法是对顶部的分隔线做分支处理，如果需要展示顶部的分隔线，就给 bounds.bottom 赋值为 child.getTop()，给 bounds.top 赋值为 child.getTop()-dividerHeight。

幸好 ListView 的这些毛病都是小问题，不影响将其发扬光大。上一章的实战项目——购物车中有商品列表展示，当时采取的是多个 LinearLayout 依次从上往下排列，在每行线性布局中再放入商品图片、名称、价格等信息，该做法在代码中动态添加每个控件，费时费力而且容易出错。这种情况用 ListView 通过适配器显示商品列表更合理，具体的代码实现过程与 BaseAdapter 方式类似，完成后的购物车页面效果如图 5-9 所示。



图 5-9　使用列表视图改造后的购物车页面

在实战中，ListView 表现得还不是很完美，有 3 个地方要特别注意：

（1）如果 ListView 下面还有其他控件，就要将 ListView 的高度设为 0dp，权重设为 1，确保列表视图扩展到所有剩余页面；如果 ListView 的高度设置为 wrap_content，系统就只预留一行高度，如此一来只有第一行显示，这显然不是我们所期望的。在图 5-9 中，我们看到结算行位于页面底部，就是因为列表视图占据了页面的剩余空间，导致结算行被挤到最下面了。

（2）给列表项注册上下文菜单也不容易，如果按照之前对上下文菜单的操作，长按列表项时 App 就会异常退出。这是因为上下文菜单的长按事件与列表项的长按监听器 OnItemLongClickListener 相互影响，使得程序陷入了死循环。最后的处理办法是要把两种长按事件阻隔开，即列表项长按事件处理完毕后才触发上下文菜单事件，打开上下文菜单之前得清空列表项的长按事件，具体代码如下：

```
    private View mCurrentView;
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view, int position, long id) {
        mCurrentGood = mCartArray.get(position);
        mCurrentView = view;
```

```
            mHandler.postDelayed(mPopupMenu, 100);
            return true;
        }

        private Handler mHandler = new Handler();
        private Runnable mPopupMenu = new Runnable() {
            @Override
            public void run() {
                lv_cart.setOnItemLongClickListener(null);
                registerForContextMenu(mCurrentView);
                openContextMenu(mCurrentView);
                unregisterForContextMenu(mCurrentView);
                lv_cart.setOnItemLongClickListener(ShoppingCartActivity.this);
            }
        };
```

（3）如果列表项包含 EditText、Button（包括 ImageButton、CheckBox 等按钮）等控件，此时点击列表项不会响应点击监听器 OnItemClickListener。罪魁祸首还是焦点抢占问题，之前介绍 EditText 时提到页面会自动弹出软键盘，就是 EditText 抢占焦点造成的。同理，列表项中如果存在 EditText 和 Button，这些子控件也会抢占列表项的焦点，使得点击操作被视为对 EditText 和 Button 的点击（无论点击处是否落在 EditText 和 Button 的范围内），而不是列表项的点击。解决办法是给列表项布局文件的根节点加上 descendantFocusability 属性，并声明在列表项范围内剥夺子控件的抢占权利，具体的属性设置代码如下：

```
    android:descendantFocusability="blocksDescendants"
```

5.2.3 网格视图 GridView

除了列表视图，网格视图 GridView 也是常见的适配器视图，用于分行分列显示表格信息，比 ListView 更适合展示商品清单。GridView 新增的属性与方法说明见表 5-2。

表 5-2 GriView 的属性与方法说明

XML 中的属性	GridView 类的设置方法	说明
horizontalSpacing	setHorizontalSpacing	指定网格项在水平方向的间距
verticalSpacing	setVerticalSpacing	指定网格项在垂直方向的间距
numColumns	setNumColumns	指定列的数目
stretchMode	setStretchMode	指定剩余空间的拉伸模式。拉伸模式的取值说明见表 5-3
columnWidth	setColumnWidth	指定每列的宽度。拉伸模式为 spacingWidth、spacingWidthUniform 时，必须指定列宽

表 5-3 拉伸模式的取值说明

XML 中的拉伸模式	GridView 类的拉伸模式	说明
none	NO_STRETCH	不拉伸
columnWidth	STRETCH_COLUMN_WIDTH	若有剩余空间，则拉伸列宽挤掉空隙



（续表）

XML 中的拉伸模式	GridView 类的拉伸模式	说明
spacingWidth	STRETCH_SPACING	若有剩余空间，则列宽不变，把空间分配到每列间的空隙
spacingWidthUniform	STRETCH_SPACING_UNIFORM	若有剩余空间，则列宽不变，把空间分配到每列左右的空隙

另外，GridView 实现了 3 个与适配器相关的方法。

- setAdapter：设置网格项的数据适配器，适配器一般继承 BaseAdapter。
- setOnItemClickListener：设置网格项的点击事件监听器，用法同 ListView。
- setOnItemLongClickListener：设置网格项的长按事件监听器，用法同 ListView。

可以看到，网格视图不像列表视图那样有指定分隔线的方法，但这并不意味着 GridView 就没法设置分隔线。通过变通的方式也能给 GridView 设置分隔线。具体地说，就是先给 GridView 设置背景色（例如黑色），以及网格之间的水平间距和垂直间距；然后给网格项设置背景色（例如白色），这样只有网格间距是黑色，从而间接设置了黑色的分隔线。

下面是演示网格视图分隔线的测试代码片段：

```
private class DividerSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        gv_planet.setBackgroundColor(Color.RED);
        gv_planet.setHorizontalSpacing(dividerPad);
        gv_planet.setVerticalSpacing(dividerPad);
        gv_planet.setStretchMode(GridView.STRETCH_COLUMN_WIDTH);
        gv_planet.setColumnWidth(250);
        gv_planet.setPadding(0, 0, 0, 0);
        if (arg2 == 0) {  // 不显示分隔线
            gv_planet.setBackgroundColor(Color.WHITE);
            gv_planet.setHorizontalSpacing(0);
            gv_planet.setVerticalSpacing(0);
        } else if (arg2 == 1) {  // 只显示内部分隔线(NO_STRETCH)
            gv_planet.setStretchMode(GridView.NO_STRETCH);
        } else if (arg2 == 2) {  // 只显示内部分隔线(COLUMN_WIDTH)
            gv_planet.setStretchMode(GridView.STRETCH_COLUMN_WIDTH);
        } else if (arg2 == 3) {  // 只显示内部分隔线(STRETCH_SPACING)
            gv_planet.setStretchMode(GridView.STRETCH_SPACING);
        } else if (arg2 == 4) {  // 只显示内部分隔线(SPACING_UNIFORM)
            gv_planet.setStretchMode(GridView.STRETCH_SPACING_UNIFORM);
        } else if (arg2 == 5) {  // 显示全部分隔线（使用 padding）
            gv_planet.setPadding(dividerPad, dividerPad, dividerPad, dividerPad);
        }
    }
```

```
            public void onNothingSelected(AdapterView<?> arg0) {
            }
        }
```

接下来观察分隔线的测试效果，如图 5-10 所示。默认情况下，网格视图没有分隔线；但通过给整个视图与网格项分别设置背景色可间接实现分隔线，如图 5-11 所示。

图 5-11 所示的分隔线是在拉伸模式为 columnWidth 时的效果，这也是最常用的拉伸模式。如果拉伸模式为其他值，间距效果就大不一样。图 5-12 所示是拉伸模式为 none 时的界面，每行右边都多出了空隙。拉伸模式为 spacingWidth 时，空隙均匀分配给每列之间的间距，即变相拉大了 horizontalSpacing，具体效果如图 5-13 所示。

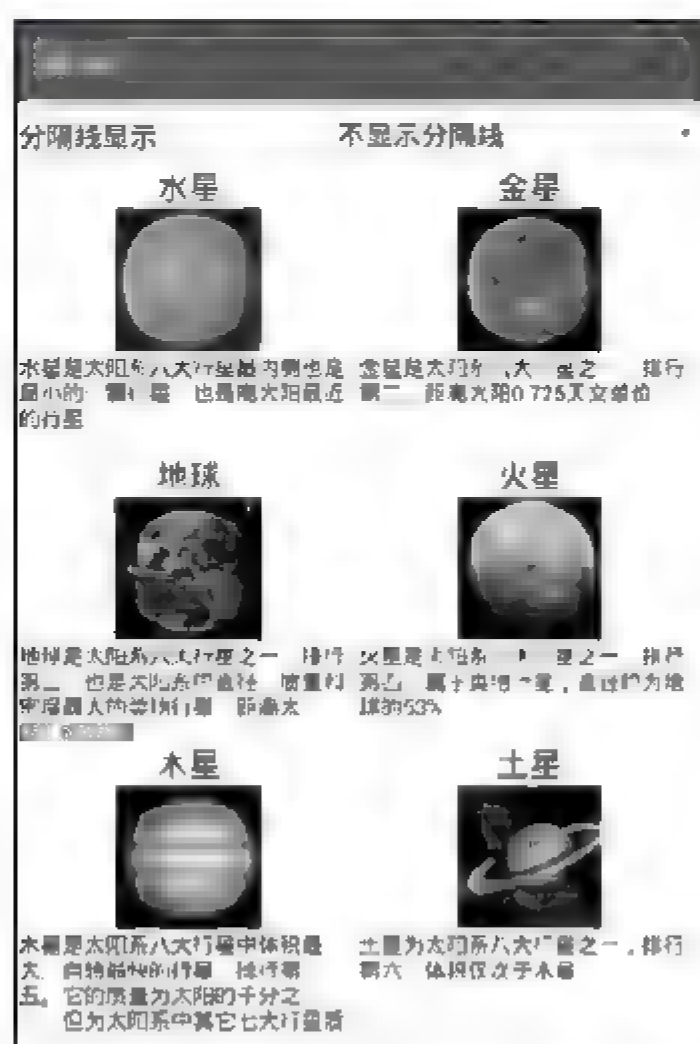


图 5-10　没有分隔线效果

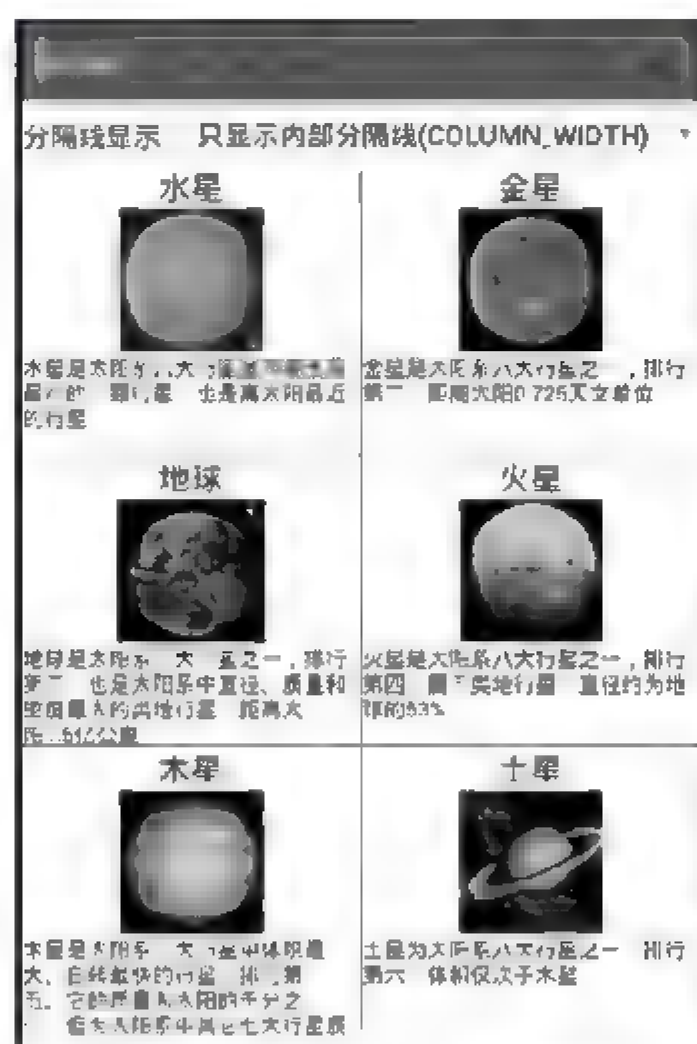


图 5-11　拉伸模式为 columnWidth

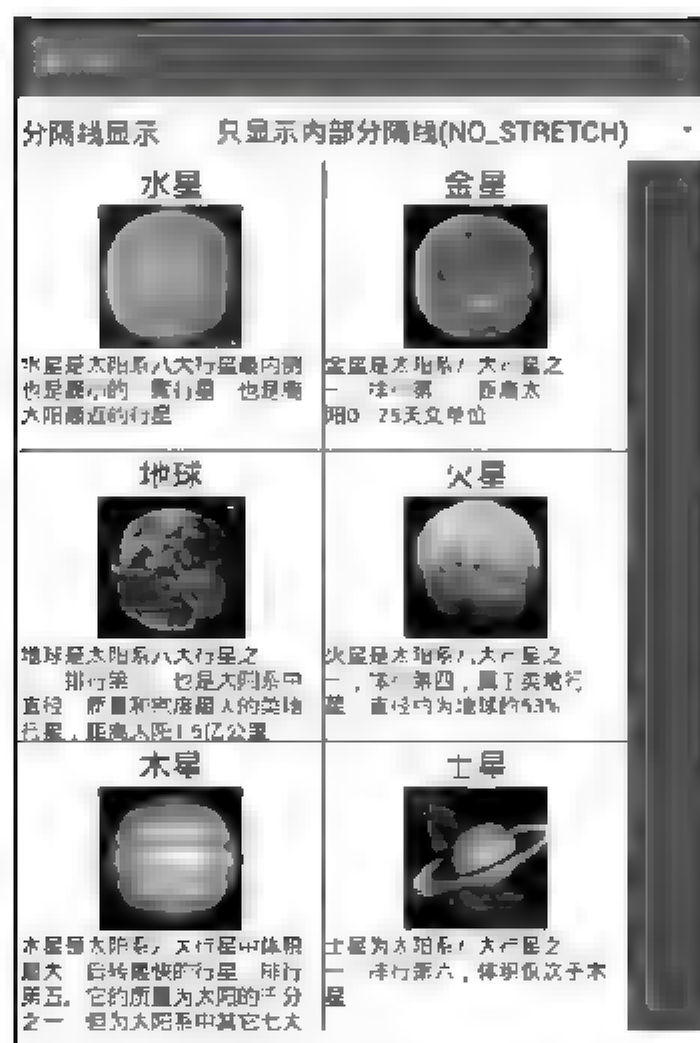


图 5-12　拉伸模式为 none

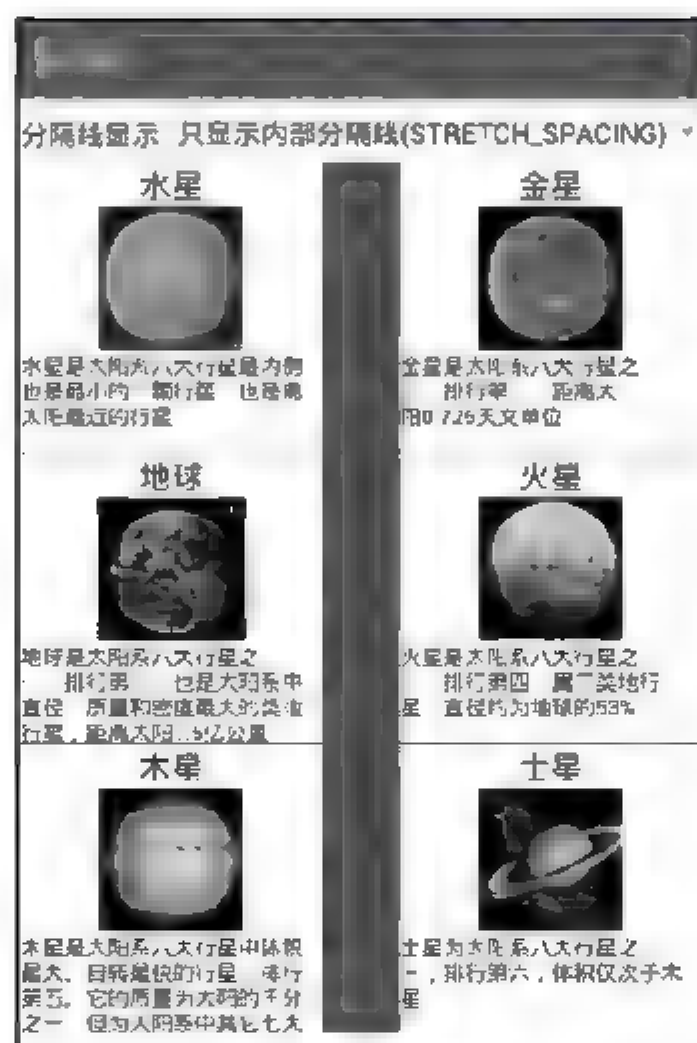


图 5-13　拉伸模式为 spacingWidth

拉伸模式为 spacingWidthUniform 时，分配给每列的空隙被分成两半，一半加到网格项的左边，一半加到网格项的右边，具体效果如图 5-14 所示。这样看来，还是 columnWidth 的拉伸最符合实际，因为不浪费空间。然而 GridView 的间距设置跟 ListView 有同样的毛病，无论是 horizontalSpacing 还是 verticalSpacing，都设置不了整个网格视图的边缘，也就是对四周的分隔线依然无能为力。这时还是得使出 padding，使用 padding 能对付不同的对象，这才能体现其精妙所在。图 5-15 所示为运用 padding 后的效果图。

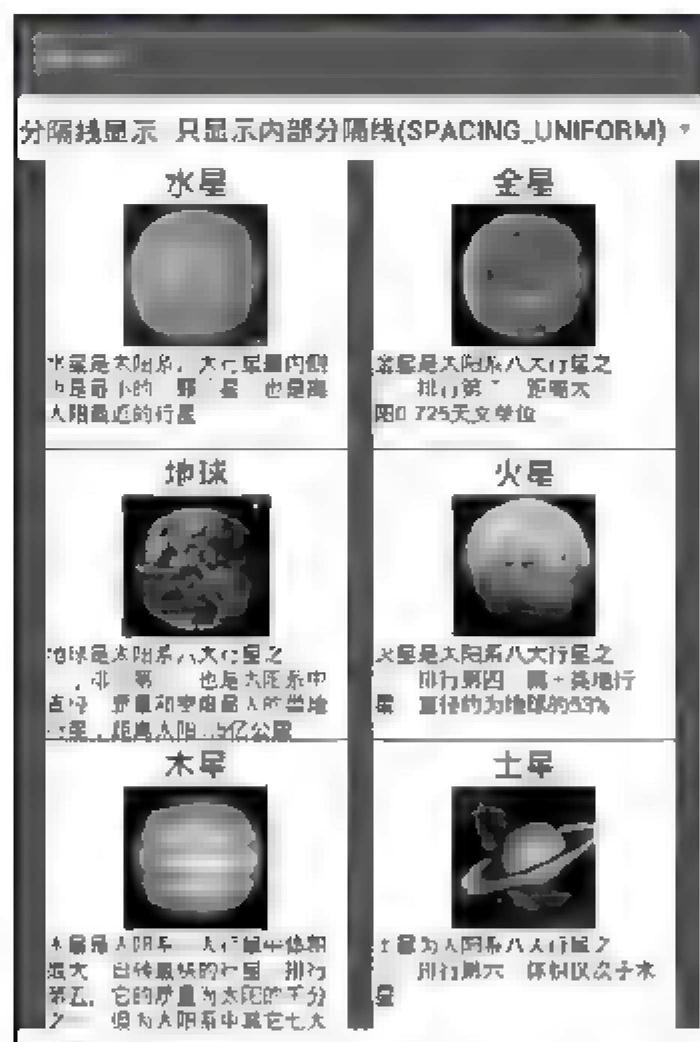


图 5-14 拉伸模式为 spacingWidthUniform

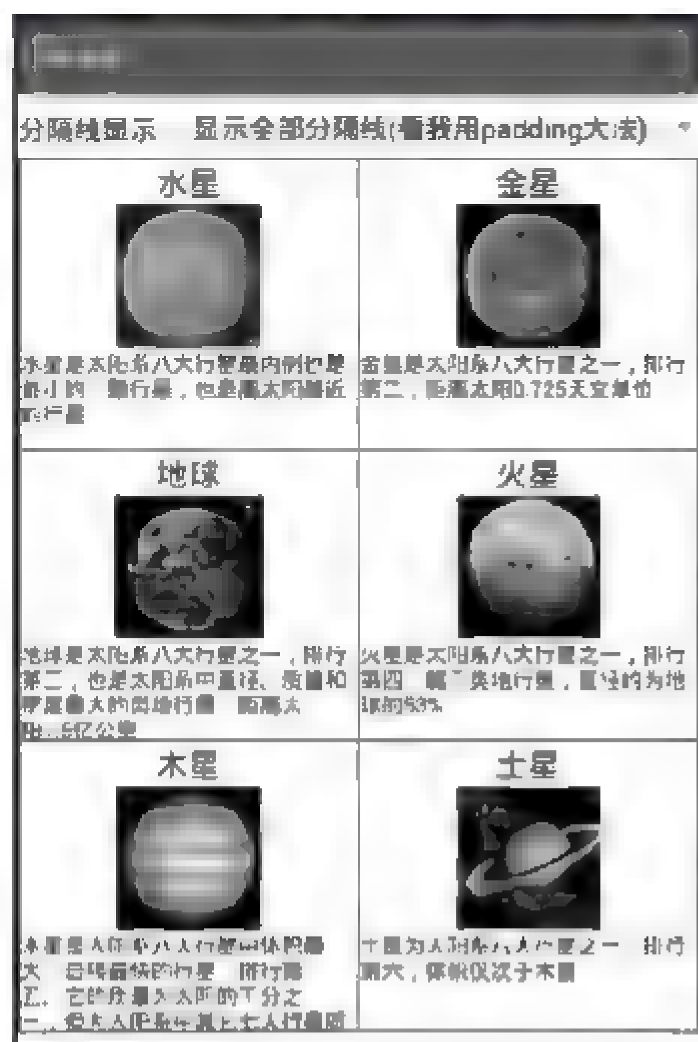


图 5-15 padding 显示四周分隔线

接下来我们继续在实战中运用 GridView，因为实践出真知。上一节的列表视图已经成功改造了购物车的商品列表，现在用网格视图改造商品频道页面，六部手机正好做成三行两列的 GridView。采用网格视图改造的商品频道页面效果如图 5-16 所示。



图 5-16 使用网格视图改造后的商品频道页面

对该页面进行功能测试时，可能会发现以下问题：

（1）网格项内有一个“加入购物车”按钮，使得网格项的点击事件失效（原本点击网格项跳转到商品详情页面）。这个问题好办，前面介绍 ListView 时已经提到了，原因是网格项的焦点被按钮抢占了，解决办法是在网格项布局的根节点加上下面这行：

```
android:descendantFocusability="blocksDescendants"
```

（2）点击“加入购物车”按钮，除了修改数据库外，还得刷新页面右上方购物车图标上的数字，相当于适配器把消息传回给 Activity。对于这个问题，可借鉴点击监听器的做法，具体步骤如下：

01 定义一个监听器接口 addCartListener，在适配器的构造函数中传入该监听器的对象，示例代码如下：

```
public GoodsAdapter(Context context, ArrayList<GoodsInfo> goods_list, addCartListener listener) {
        mInflater = LayoutInflater.from(context);
        mContext = context;
        mGoodsArray = goods_list;
        mAddCartListener = listener;
}

private addCartListener mAddCartListener;
public static interface addCartListener {
        public void addToCart(long goods_id);
}
```

02 使用适配器处理“加入购物车”按钮的点击操作时，调用监听器的内部方法 addToCart，示例代码如下：

```
        mAddCartListener.addToCart(info.rowid);
```

03 Activity 的页面代码要实现 addCartListener 接口的 addToCart 方法，进行对应的购物车业务逻辑处理，同时记住往适配器的构造函数传入该监听器的对象。

5.3 翻页类视图

本节介绍如何在页面上运用翻页类视图，包括翻页视图 ViewPager 配合翻页适配器 PagerAdapter 的用法、翻页标题栏 PagerTitleStrip/PagerTabStrip 的用法，最后结合实战演示使用 ViewPager 实现简单的启动引导页效果。

5.3.1 翻页视图 ViewPager

上一节介绍的 ListView 与 GridView，一个分行展示，另一个分行又分列，其实都是在垂直方向上下滑动。有没有一种控件允许页面在水平方向左右滑动，就像翻书、翻报纸一样呢？对于这种左右滑动的翻页功能，Android 提供了已经封装好的控件，就是翻页视图 ViewPager。对于 ViewPager 来说，一个页面就是一个项（相当于 ListView 的一个列表项），许多页面组成 ViewPager 的页面项。

明确了 ViewPager 的原理类似 ListView 和 GridView，翻页视图的用法也与之类似。ListView 和 GridView 的适配器使用 BaseAdapter，ViewPager 的适配器使用 PagerAdapter；ListView 和 GridView 的监听器使用 OnItemClickListener，ViewPager 的监听器使用 OnPageChangeListener，表示监听页面切换事件。

下面是 ViewPager 三个常用方法的说明。

- setAdapter：设置页面项的适配器。适配器用的是 PagerAdapter 及其子类。
- setCurrentItem：设置当前页码，即打开翻页视图时默认显示哪个页面。
- addOnPageChangeListener：设置翻页视图的页面切换监听器。该监听器需实现接口 OnPageChangeListener 下的 3 个方法，具体说明如下。
  - onPageScrollStateChanged：在页面滑动状态变化时触发。
  - onPageScrolled：在页面滑动过程中触发。
  - onPageSelected：在选中页面时，即滑动结束后触发。

翻页适配器 PagerAdapter 与基本适配器 BaseAdapter 的用法相近，需实现构造函数、获取页面个数的 getCount 方法、生成单个页面视图的 instantiateItem 方法，另外多了一个回收页面的 destroyItem 方法。下面是使用 PagerAdapter 的代码：

```
public class ImagePagerAdapater extends PagerAdapter {
    private Context mContext;
    private ArrayList<ImageView> mViewList = new ArrayList<ImageView>();
    private ArrayList<GoodsInfo> mGoodsList = new ArrayList<GoodsInfo>();

    public ImagePagerAdapater(Context context, ArrayList<GoodsInfo> goodsList) {
        mContext = context;
        mGoodsList = goodsList;
        for (int i=0; i<mGoodsList.size(); i++) {
            ImageView view = new ImageView(mContext);
            view.setLayoutParams(new LayoutParams(
                    LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
            view.setImageResource(mGoodsList.get(i).pic);
            view.setScaleType(ScaleType.FIT_CENTER);
            mViewList.add(view);
        }
    }

    @Override
    public int getCount() {
        return mViewList.size();
    }

    @Override
    public boolean isViewFromObject(View arg0, Object arg1) {
        return arg0 == arg1;
    }

    @Override
    public void destroyItem(ViewGroup container, int position, Object object) {
        container.removeView(mViewList.get(position));
```

```
        }

        @Override
        public Object instantiateItem(ViewGroup container, int position) {
            container.addView(mViewList.get(position));
            return mViewList.get(position);
        }
    }
```

与适配器 ImagePagerAdapater 对应的页面代码如下：

```
    public class ViewPagerActivity extends AppCompatActivity implements OnPageChangeListener {
        private ArrayList<GoodsInfo> goodsList;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_view_pager);
            ViewPager vp_content = (ViewPager) findViewById(R.id.vp_content);
            goodsList = GoodsInfo.getDefaultList();
            ImagePagerAdapater adapter = new ImagePagerAdapater(this, goodsList);
            vp_content.setAdapter(adapter);
            vp_content.setCurrentItem(0);
            vp_content.addOnPageChangeListener(this);
        }

        //翻页状态改变时调用，状态参数取值说明为：0 表示静止，1 表示正在滑动，2 表示滑动完毕
        //在翻页过程中，状态值变化依次为：正在滑动→滑动完毕→静止
        @Override
        public void onPageScrollStateChanged(int arg0) {
        }

        //在翻页过程中调用。该方法的三个参数取值说明为：第一个参数表示当前页面的序号
        //第二个参数表示当前页面偏移的百分比，取值为 0 到 1；第三个参数表示当前页面的偏移距离
        @Override
        public void onPageScrolled(int arg0, float arg1, int arg2) {
        }

        @Override
        public void onPageSelected(int arg0) {
            Toast.makeText(this, "您翻到的手机品牌是：" +goodsList.get(arg0).name,
Toast.LENGTH_SHORT).show();
        }
    }
```

下面是页面代码对应的布局文件代码，注意 ViewPager 的节点名必须引用 v4 包的全路径，即 android.support.v4.view.ViewPager。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >
    <android.support.v4.view.ViewPager
        android:id="@+id/vp_content"
        android:layout_width="match_parent"
        android:layout_height="400dp" />
</LinearLayout>
```

具体的翻页效果如图 5-17 所示。截图的瞬间，ViewPager 正在左右两个页面之间滑动。



图 5-17　翻页视图滚动瞬间

5.3.2　翻页标题栏 PagerTitleStrip/PagerTabStrip

为了方便开发者处理 ViewPager 的页码显示与切换，Android 附带提供了两个控件，分别是 PagerTitleStrip 和 PagerTabStrip。二者都是在 ViewPager 页面上方展示设定的页面标题，不同之处在于 PagerTitleStrip 只是单纯的文本标题效果，无法点击进行页面切换；PagerTabStrip 类似选项卡效果，文本下面有横线，点击左右选项卡即可切换到对应页面。要想在标题栏显示指定的文字，得重写 PagerAdapter 的 getPageTitle 方法，在这方面两个控件的处理是一样的，示例代码如下：

```
@Override
public CharSequence getPageTitle(int position) {
    return mGoodsList.get(position).name;
}
```

下面是在布局文件中添加 PagerTitleStrip 的代码，注意 PagerTitleStrip 的节点名必须引用 v4 包的全路径，即 android.support.v4.view.PagerTitleStrip。如果用 PagerTabStrip，就把

PagerTitleStrip 改为 PagerTabStrip。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <android.support.v4.view.ViewPager
        android:id="@+id/vp_content"
        android:layout_width="match_parent"
        android:layout_height="400dp" >

        <android.support.v4.view.PagerTitleStrip
            android:id="@+id/pts_title"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </android.support.v4.view.ViewPager>
</LinearLayout>
```

翻页标题栏的显示界面很简单，正上方是当前页面的标题，左上方是左边页面的标题，右上方是右边页面的标题。PagerTitleStrip 的标题只有文字，如图 5-18 所示。PagerTabStrip 除了文字还有下划线，如图 5-19 所示。



图 5-18　PagerTitleStrip 的效果



图 5-19　PagerTabStrip 的效果

标题栏因为只有文本，所以调整样式只能改改文字的大小与颜色。注意这两个控件没法在布局文件中修改文字样式，因为没有对应的样式属性，只能在代码中调用文本样式的设置方法，具体的代码如下：

```
PagerTabStrip pts_tab = (PagerTabStrip) findViewById(R.id.pts_tab);
pts_tab.setTextSize(TypedValue.COMPLEX_UNIT_SP, 20);
pts_tab.setTextColor(Color.GREEN);
```

5.3.3 简单的启动引导页

ViewPager 的应用很广，当我们安装一个新的 App 时，第一次启动大多出现欢迎页面，这个引导页通常要往右翻好几页，才会进入 App 的主页面。启动引导页的效果大多是 ViewPager 做的。

下面就来动手打造你的第一个 App 启动欢迎页吧！ViewPager 技术的核心在于页面项的布局及其适配器，因此首先要设计页面项的布局。一般来说，引导页主要由两部分组成，一部分是背景图；另一部分是页面下方的一排圆点，高亮的圆点表示当前位于第几页。具体效果如图 5-20 与图 5-21 所示。其中，图 5-20 所示为欢迎页面的第一页；图 5-21 所示为第二页，高亮圆点移到第二个。



图 5-20　欢迎页的第一页

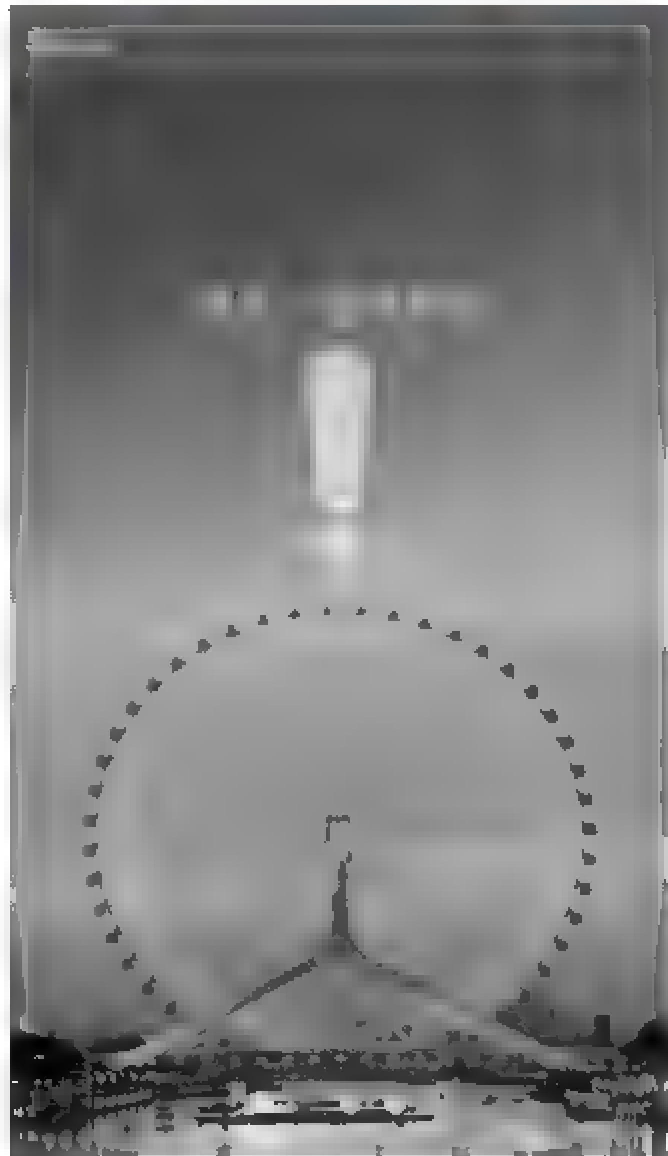


图 5-21　欢迎页的第二页

除了背景图与一排圆点，最后一页往往有一个按钮，是进入主页面的入口。页面项的布局文件至少有 3 个控件：背景图（采用 ImageView）、一排圆点（可采用 RadioGroup）、入口按钮（采用 Button），详细的代码如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <ImageView
        android:id="@+id/iv_launch"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY" />
```

```
    <RadioGroup
        android:id="@+id/rg_indicate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_gravity="bottom|center"
        android:orientation="horizontal"
        android:paddingBottom="20dp" />

    <Button
        android:id="@+id/btn_start"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="80dp"
        android:layout_marginRight="80dp"
        android:layout_centerInParent="true"
        android:gravity="center"
        android:text="立即开始美好生活"
        android:textColor="#ff3300"
        android:textSize="22sp"
        android:visibility="gone" />"
</RelativeLayout>
```

根据该布局文件，引导页的最后两个页面如图 5-22 与图 5-23 所示。其中，图 5-22 是第 3 个页面，高亮圆点移到第 3 个；图 5-23 是最后一个页面，只有该页才会显示入口按钮。



图 5-22　欢迎页的第三页

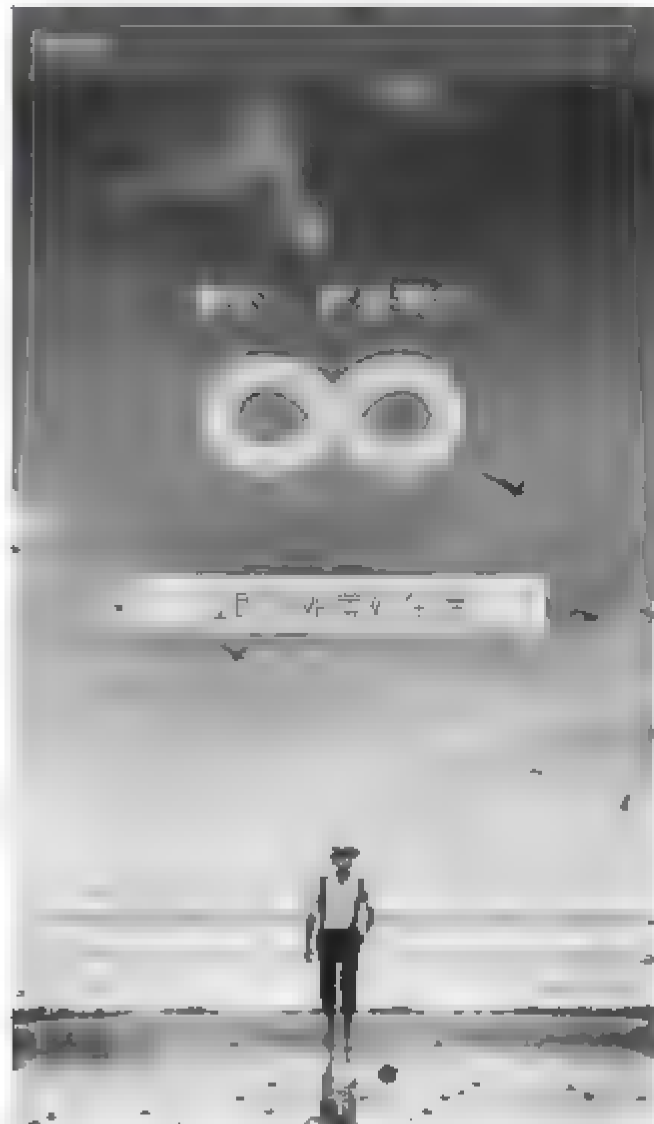


图 5-23　欢迎页的最后一页

启动引导页的适配器代码主要工作是根据布局文件构造每页的视图，然后把当前页码的圆点设置高亮，如果是最后一页就显示入口按钮，具体代码如下：

```
    public class LaunchSimpleAdapter extends PagerAdapter {
        private LayoutInflater mInflater;
        private Context mContext;
        private ArrayList<View> mViewList = new ArrayList<View>();

        public LaunchSimpleAdapter(Context context, int[] imageArray) {
            mInflater = LayoutInflater.from(context);
            mContext = context;
            for (int i=0; i<imageArray.length; i++) {
                View view = mInflater.inflate(R.layout.item_launch, null);
                ImageView iv_launch = (ImageView) view.findViewById(R.id.iv_launch);
                RadioGroup rg_indicate = (RadioGroup) view.findViewById(R.id.rg_indicate);
                Button btn_start = (Button) view.findViewById(R.id.btn_start);
                iv_launch.setImageResource(imageArray[i]);
                for (int j=0; j<imageArray.length; j++) {
                    RadioButton radio = new RadioButton(mContext);
                    radio.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT));
                    radio.setButtonDrawable(R.drawable.launch_guide);
                    radio.setPadding(10, 10, 10, 10);
                    rg_indicate.addView(radio);
                }
                ((RadioButton)rg_indicate.getChildAt(i)).setChecked(true);
                if (i == imageArray.length-1) {
                    btn_start.setVisibility(View.VISIBLE);
                    btn_start.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                            Toast.makeText(mContext, "欢迎您开启美好生活",
Toast.LENGTH_SHORT).show();
                        }
                    });
                }
                mViewList.add(view);
            }
        }

        @Override
        public int getCount() {
            return mViewList.size();
```

```
        }

        @Override
        public boolean isViewFromObject(View arg0, Object arg1) {
            return arg0 == arg1;
        }

        @Override
        public void destroyItem(ViewGroup container, int position, Object object) {
            container.removeView(mViewList.get(position));
        }

        @Override
        public Object instantiateItem(ViewGroup container, int position) {
            container.addView(mViewList.get(position));
            return mViewList.get(position);
        }
    }
```

5.4 碎片 Fragment

本节介绍如何在页面上加入碎片并合理使用，包括通过静态注册方式使用碎片 Fragment、通过动态注册方式配合碎片适配器 FragmentStatePagerAdapter 使用 Fragment，并分别分析两种注册方式的 Fragment 生命周期，最后结合实战使用 Fragment 对启动引导页进行改进。

5.4.1 静态注册

Fragment 是个特别的存在，有点像报纸上的专栏，看起来只占据页面的一小块，但是这一小块有自己的生命周期，可以自行其事，仿佛独立王国；并且这一小块的特性无论在哪个页面，给一个位置就行，添加后不影响宿主页面的其他区域，去除后也不影响宿主页面的其他区域。

每个 Fragment 都有对应的布局文件，依据其使用方式可分为静态注册与动态注册两类。静态注册是在布局文件中直接放置 fragment 节点，类似于一个普通控件，可被多个布局文件同时引用。静态注册一般用于某个通用的页面部件（如 Logo 条、广告条等），每个活动页面均可直接引用该部件。

下面是 Fragment 布局文件的代码，看起来跟列表项与网格项的布局文件差不多。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:background="#bbffbb" >
```



```
    <TextView
        android:id="@+id/tv_adv"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:text="广告"
        android:textColor="#000000"
        android:textSize="17sp" />

    <ImageView
        android:id="@+id/iv_adv"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="5"
        android:src="@drawable/adv"
        android:scaleType="fitCenter" />
</LinearLayout>
```

下面是与上述布局对应的 Fragment 代码，除了继承自 Fragment 外，其他地方很像活动页面代码。

```
public class StaticFragment extends Fragment implements OnClickListener {
    protected View mView;
    protected Context mContext;
    private TextView tv_adv;
    private ImageView iv_adv;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        mContext = getActivity();
        mView = inflater.inflate(R.layout.fragment_static, container, false);
        tv_adv = (TextView) mView.findViewById(R.id.tv_adv);
        iv_adv = (ImageView) mView.findViewById(R.id.iv_adv);
        tv_adv.setOnClickListener(this);
        iv_adv.setOnClickListener(this);
        return mView;
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.tv_adv) {
            Toast.makeText(mContext, "您点击了广告文本", Toast.LENGTH_LONG).show();
```

```
        } else if (v.getId() == R.id.iv_adv) {
            Toast.makeText(mContext, "您点击了广告图片", Toast.LENGTH_LONG).show();
        }
    }
}
```

若想在页面布局文件中引用 Fragment，则可直接加入一个 fragment 节点，注意 fragment 节点要增加 name 属性指定该 Fragment 类的完整路径。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp" >

    <fragment
        android:id="@+id/fragment_static"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:name="com.example.senior.fragment.StaticFragment" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center|top"
        android:text="这里是每个页面的具体内容"
        android:textColor="#000000"
        android:textSize="17sp" />
</LinearLayout>
```

最后运行并查看页面效果，如图 5-24 所示。此时 Fragment 界面给人的感觉就像一个视图，同样可以接收点击事件。



图 5-24　静态注册的 Fragment 效果

使用静态注册需要注意以下两点：

（1）fragment 节点必须指定 id 属性，否则 App 运行时会报错 Must specify unique android:id, android:tag, or have a parent with an id for ***。



（2）如果页面代码继承自 Activity，Fragment 类就必须继承自 android.app.Fragment，不能使用 android.support.v4.app.Fragment，否则 App 运行会报错 Trying to instantiate a class *** that is not a Fragment 或报错 java.lang.ClassCastException：*** cannot be cast to android.app.Fragment；如果页面代码继承自 AppCompatActivity 或 FragmentActivity，那么无论是 android.app.Fragment 还是 android.support.v4.app.Fragment 都可以使用。

另外，介绍一下 Fragment 在静态注册时的生命周期，如 Activity 的基本生命周期方法 onCreate、onStart、onResume、onPause、onStop、onDestroy，碎片 Fragment 都有，而且还多出了下面 5 个生命周期方法。

- onAttach：与 Activity 结合。可在该方法中实例化 Activity 的一个回调对象，在 Fragment 中调用 Activity 的回调方法。这样设计的好处是 Activity 无须调用 set***Listener 方法设置监听器接口。
- onCreateView：创建碎片视图。
- onActivityCreated：在活动页面创建完毕后调用。
- onDestroyView：回收碎片视图。
- onDetach：与 Activity 分离。

至于这些周期方法的先后调用顺序，观察日志最简单明了。下面是打开页面时的日志信息，此时 Fragment 的 onCreate 操作先于 Activity，而 onStart 与 onResume 操作在 Activity 之后。

```
12:26:11.506：D/StaticFragment(5809)：onAttach
12:26:11.506：D/StaticFragment(5809)：onCreate
12:26:11.530：D/StaticFragment(5809)：onCreateView
12:26:11.530：D/FragmentStaticActivity(5809)：onCreate
12:26:11.530：D/StaticFragment(5809)：onActivityCreated
12:26:11.530：D/FragmentStaticActivity(5809)：onStart
12:26:11.530：D/StaticFragment(5809)：onStart
12:26:11.530：D/FragmentStaticActivity(5809)：onResume
12:26:11.530：D/StaticFragment(5809)：onResume
```

下面是退出页面时的日志信息，此时 Fragment 的 onPause、onStop、onDestroy 都在 Activity 之前。

```
12:26:36.586：D/StaticFragment(5809)：onPause
12:26:36.586：D/FragmentStaticActivity(5809)：onPause
12:26:36.990：D/StaticFragment(5809)：onStop
12:26:36.990：D/FragmentStaticActivity(5809)：onStop
12:26:36.990：D/StaticFragment(5809)：onDestroyView
12:26:36.990：D/StaticFragment(5809)：onDestroy
12:26:36.990：D/StaticFragment(5809)：onDetach
12:26:36.990：D/FragmentStaticActivity(5809)：onDestroy
```

总结一下，在静态注册时，除了碎片的创建操作在页面创建之前，其他操作都在页面创建之后。就像老实本分的下级，上级开腔后才能说话，上级要做总结性发言时赶紧闭嘴。



5.4.2 动态注册/碎片适配器 FragmentStatePagerAdapter

Fragment 拥有两种使用方式，即静态注册和动态注册。相比静态注册，实际开发中动态注册用的更多。静态注册在布局文件中直接指定 Fragment，而动态注册直到在代码中才动态添加 Fragment。动态生成的碎片给谁用、要怎么用呢？毫无疑问，动态碎片就是给翻页视图用的，ViewPager 和 Fragment 是一对好搭档。

要说怎么在 ViewPager 中使用 Fragment，关键在于适配器。上一节演示 ViewPager 时用的适配器是翻页适配器 PagerAdapter。如果结合 Fragment，适配器就要改用碎片适配器 FragmentStatePagerAdapter。下面是使用 FragmentStatePagerAdapter 适配器的代码，获取页面视图的地方变成了 getItem 方法。

```
public class MobilePagerAdapter extends FragmentStatePagerAdapter {
    private ArrayList<GoodsInfo> mGoodsList = new ArrayList<GoodsInfo>();
    public MobilePagerAdapter(FragmentManager fm, ArrayList<GoodsInfo> goodsList) {
        super(fm);
        mGoodsList = goodsList;
    }

    public int getCount() {
        return mGoodsList.size();
    }

    public Fragment getItem(int position) {
        return DynamicFragment.newInstance(position,
                mGoodsList.get(position).pic, mGoodsList.get(position).desc);
    }

    @Override
    public CharSequence getPageTitle(int position) {
        return mGoodsList.get(position).name;
    }
}
```

以上适配器在获得碎片对象时不用构造函数，却用了 newInstance 方法，目的是给 Fragment 传递参数信息。通过构造函数获得碎片对象后还得调用 setArguments 方法才能把请求数据塞进去，然后在 Fragment 的 onCreateView 函数中调用 getArguments 方获得请求数据。下面是动态注册的碎片代码：

```
public class DynamicFragment extends Fragment {
    protected View mView;
    protected Context mContext;
    private int mPosition;
    private int mImageId;
```

```
        private String mDesc;

        public static DynamicFragment newInstance(int position, int image_id, String desc) {
            DynamicFragment fragment = new DynamicFragment();
            Bundle bundle = new Bundle();
            bundle.putInt("position", position);
            bundle.putInt("image_id", image_id);
            bundle.putString("desc", desc);
            fragment.setArguments(bundle);
            return fragment;
        }

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
            mContext = getActivity();
            if (getArguments() != null) {
                mPosition = getArguments().getInt("position", 0);
                mImageId = getArguments().getInt("image_id", 0);
                mDesc = getArguments().getString("desc");
            }
            mView = inflater.inflate(R.layout.fragment_dynamic, container, false);
            ImageView iv_pic = (ImageView) mView.findViewById(R.id.iv_pic);
            TextView tv_desc = (TextView) mView.findViewById(R.id.tv_desc);
            iv_pic.setImageResource(mImageId);
            tv_desc.setText(mDesc);
            return mView;
        }
    }
```

现在有了适用于动态注册的适配器与碎片对象，还需要一个主页面配合才能完成整个页面的展示。下面是动态注册用到的页面代码，注意这里不能继承 Activity，只能继承 AppCompatActivity 或 FragmentActivity。

```
    public class FragmentDynamicActivity extends FragmentActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_fragment_dynamic);
            PagerTabStrip pts_tab = (PagerTabStrip) findViewById(R.id.pts_tab);
            pts_tab.setTextSize(TypedValue.COMPLEX_UNIT_SP, 20);
            ViewPager vp_content = (ViewPager) findViewById(R.id.vp_content);
            ArrayList<GoodsInfo> goodsList = GoodsInfo.getDefaultList();
            MobilePagerAdapter adapter = new MobilePagerAdapter(getSupportFragmentManager(),
goodsList);
```

```
            vp_content.setAdapter(adapter);
            vp_content.setCurrentItem(0);
        }
    }
```

运行效果如图 5-25 所示，看起来 Fragment 的界面与上一节 ViewPager 的效果没什么不同。



图 5-25　动态注册的 Fragment 效果

下面来看 Fragment 的生命周期。惯例先输出代码加上生命周期的日志，然后观察动态注册的运行日志。下面是打开页面时的日志信息：

```
12:28:28.074: D/FragmentDynamicActivity(5809): onCreate
12:28:28.074: D/FragmentDynamicActivity(5809): onStart
12:28:28.074: D/FragmentDynamicActivity(5809): onResume
12:28:28.086: D/DynamicFragment(5809): onAttach position=0
12:28:28.086: D/DynamicFragment(5809): onCreate position=0
12:28:28.114: D/DynamicFragment(5809): onCreateView position=0
12:28:28.114: D/DynamicFragment(5809): onActivityCreated position=0
12:28:28.114: D/DynamicFragment(5809): onStart position=0
12:28:28.114: D/DynamicFragment(5809): onResume position=0
12:28:28.114: D/DynamicFragment(5809): onAttach position=0
12:28:28.114: D/DynamicFragment(5809): onCreate position=0
12:28:28.146: D/DynamicFragment(5809): onCreateView position=1
12:28:28.146: D/DynamicFragment(5809): onStart position=1
12:28:28.146: D/DynamicFragment(5809): onResume position=1
```

下面是退出页面时的日志信息：

```
12:28:57.994: D/DynamicFragment(5809): onPause position=0
12:28:57.994: D/DynamicFragment(5809): onPause position=1
12:28:57.994: D/FragmentDynamicActivity(5809): onPause
12:28:58.402: D/DynamicFragment(5809): onStop position=0
12:28:58.402: D/DynamicFragment(5809): onStop position=1
```



```
12:28:58.402: D/FragmentDynamicActivity(5809): onStop
12:28:58.402: D/DynamicFragment(5809): onDestroyView position=0
12:28:58.402: D/DynamicFragment(5809): onDestroy position=0
12:28:58.402: D/DynamicFragment(5809): onDetach position=0
12:28:58.402: D/DynamicFragment(5809): onDestroyView position=1
12:28:58.402: D/DynamicFragment(5809): onDestroy position=1
12:28:58.402: D/DynamicFragment(5809): onDetach position=1
12:28:58.402: D/FragmentDynamicActivity(5809): onDestroy
```

日志搜集完毕，接下来分析一下这其中的奥妙。笔者总结了一下，主要有以下三点：

（1）动态注册时，Fragment 的 onCreate 操作在 Activity 之后，其余操作的先后顺序与静态注册时保持一致。

（2）注意 onActivityCreated 方法。无论是静态注册还是动态注册，该方法都在 Activity 的 onCreate 操作之后。可见该方法在页面创建之后才调用。

（3）最重要的一点，进入第一个 Fragment，实际只加载了第一页和第二页，并没有加载全部 Fragment。这正是 Fragment 的优越之处，无论当前位于哪一页，系统都只会加载当前页及相邻的前后两页，总共加载不超过三页。一旦发生页面切换，相邻页面就被加载，非相邻页面就被回收。这么做的好处是节省了宝贵的系统资源，只有用户正在浏览与将要浏览的 Fragment 才会加载，避免所有 Fragment 一起加载造成资源浪费，这正是普通 ViewPager 的缺点。

5.4.3　改进的启动引导页

接下来把 Fragment 用于实战，为“5.3.3 简单的启动引导页”做个改进。与之前相比，布局文件不变，改动的都是代码。下面是碎片适配器的代码：

```
public class LaunchImproveAdapter extends FragmentStatePagerAdapter {
    private ArrayList<Integer> mImageList = new ArrayList<Integer>();
    public LaunchImproveAdapter(FragmentManager fm, int[] imageArray) {
        super(fm);
        for (int i=0; i<imageArray.length; i++) {
            mImageList.add(imageArray[i]);
        }
    }

    public int getCount() {
        return mImageList.size();
    }

    public Fragment getItem(int position) {
        return LaunchFragment.newInstance(position, mImageList.get(position));
    }
}
```

下面是每个启动页的 Fragment 代码：

```
public class LaunchFragment extends Fragment {
    protected View mView;
    protected Context mContext;
    private int mPosition;
    private int mImageId;
    private int mCount = 4;

    public static LaunchFragment newInstance(int position, int image_id) {
        LaunchFragment fragment = new LaunchFragment();
        Bundle bundle = new Bundle();
        bundle.putInt("position", position);
        bundle.putInt("image_id", image_id);
        fragment.setArguments(bundle);
        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        mContext = getActivity();
        if (getArguments() != null) {
            mPosition = getArguments().getInt("position", 0);
            mImageId = getArguments().getInt("image_id", 0);
        }
        mView = inflater.inflate(R.layout.item_launch, container, false);
        ImageView iv_launch = (ImageView) mView.findViewById(R.id.iv_launch);
        RadioGroup rg_indicate = (RadioGroup) mView.findViewById(R.id.rg_indicate);
        Button btn_start = (Button) mView.findViewById(R.id.btn_start);
        iv_launch.setImageResource(mImageId);
        for (int j=0; j<mCount; j++) {
            RadioButton radio = new RadioButton(mContext);
            radio.setLayoutParams(new LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT));
            radio.setButtonDrawable(R.drawable.launch_guide);
            radio.setPadding(10, 10, 10, 10);
            rg_indicate.addView(radio);
        }
        ((RadioButton)rg_indicate.getChildAt(mPosition)).setChecked(true);
        if (mPosition == mCount-1) {
            btn_start.setVisibility(View.VISIBLE);
            btn_start.setOnClickListener(new OnClickListener() {
                @Override
                public void onClick(View v) {
                    Toast.makeText(mContext, "欢迎开启美好生活",
```

```
Toast.LENGTH_SHORT).show();
                                }
                        });
                }
                return mView;
        }
}
```

改进后的引导页跟之前差不多，这里只列出最后一页的效果图，如图 5-26 所示。



图 5-26　Fragment 改造后的启动引导页

5.5　Broadcast 基础

本节介绍为何使用广播 Broadcast 和如何使用广播，包括发送临时广播、注册接收器 BroadcastReceiver 接收临时广播、通过定时器设置定时广播、在 AndroidManifest.xml 中注册接收器接收系统发出的定时广播。

5.5.1　发送/接收临时广播

页面与页面之间传递和传回消息可使用 Intent。页面向适配器传递消息可使用适配器的构造函数；适配器向页面传回消息有点麻烦，在“5.2.3 网格视图 GridView”的商品频道改造时就遇到了，当时是在适配器构造函数中传入回调接口，适配器调用回调接口的方法，从而实现把消息传回页面。页面向碎片传递消息可在碎片适配器中为碎片对象设置情景参数（调用 setArguments 方法）。碎片如何把消息传回页面呢？这个问题看起来很高深，其实至少有两种解决办法。

（1）Fragment 提供了 onAttach 方法，onAttach 方法指定了结合的 Activity 对象。同样定义一个回调接口，把 Activity 对象强制转换为回调接口就可以在碎片中调用页面方法。这种方式不是本节的重点，有兴趣的读者可以自行钻研。

（2）人人都想成为武林高手，捷径之一就是寻找武功秘笈。同样是武术教材，清风剑法练十年还不如九阴真经练一年。Android 隐藏着不少武林大法，每当你按照常规思路难以解决问题时，往往用一个大法就可以迎刃而解。“5.2 列表类视图”在处理 ListView 与 GridView 的分隔线时使用到了 padding 大法。现在适配器向页面传回消息有一个 Broadcast 大法，无论对方在何处，只要用 Broadcast 大法吼一吼，对方立刻能够听到，岂不妙哉！

广播（Broadcast）用于 Android 组件之间的灵活通信，与 Activity 的区别在于：

（1）Activity 只能一对一通信；Broadcast 可以一对多，一人发送广播，多人接收处理。

（2）对于发送者来说，广播不需要考虑接收者有没有在工作，接收者在工作就接收广播，不在工作就丢弃广播。

（3）对于接收者来说，会收到各式各样的广播，所以接收者要自行过滤符合条件的广播，才能进行解包处理。

与广播有关的方法主要有以下 3 个。

- sendBroadcast：发送广播。
- registerReceiver：注册接收器，一般在 onStart 或 onResume 方法中注册。
- unregisterReceiver：注销接收器，一般在 onStop 或 onPause 方法中注销。

如果广播是在应用内使用，不需要跨进程，建议使用 LocalBroadcastManager 下的 registerReceiver 与 unregisterReceiver 方法，因为这样不但更有效率（不需要跨进程通信），而且不用考虑广播开放造成的安全问题（如果其他应用也能收到广播）。

为说明广播的工作流程，对其进行具体的演示。现在 Fragment 内有一个 Spinner 下拉框，可选择背景颜色，一旦选中某个背景色，整个活动页面的背景色就换成新颜色。Fragment 内部发现选中颜色后，要发送一个背景色变更的广播，代码如下：

```
	public final static String EVENT = "com.example.senior.fragment.BroadcastFragment";
	private String[] mColorNameArray = {"红色", "黄色", "绿色", "青色", "蓝色"};
	private int[] mColorIdArray= {Color.RED, Color.YELLOW, Color.GREEN, Color.CYAN,
Color.BLUE};
	class ColorSelectedListener implements OnItemSelectedListener {
		public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
			Intent intent = new Intent(BroadcastFragment.EVENT);
			intent.putExtra("color", mColorIdArray[arg2]);
			LocalBroadcastManager.getInstance(mContext).sendBroadcast(intent);
		}

		public void onNothingSelected(AdapterView<?> arg0) {
		}
	}
```

同时，Activity 代码要实现背景色变更的广播接收器。一旦接收到背景色变更的广播，就立即修改页面为最新的背景色，示例代码如下：

```
@Override
public void onStart() {
    super.onStart();
    bgChangeReceiver = new BgChangeReceiver();
    IntentFilter filter = new IntentFilter(BroadcastFragment.EVENT);
    LocalBroadcastManager.getInstance(this).registerReceiver(bgChangeReceiver, filter);
}

@Override
public void onStop() {
    LocalBroadcastManager.getInstance(this).unregisterReceiver(bgChangeReceiver);
    super.onStop();
}

private BgChangeReceiver bgChangeReceiver;
private class BgChangeReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (intent != null) {
            int color = intent.getIntExtra("color", Color.WHITE);
            ll_brd_temp.setBackgroundColor(color);
        }
    }
}
```

广播效果如图 5-27 所示。在 Fragment 内部选择青色，整个页面的背景色都变了。

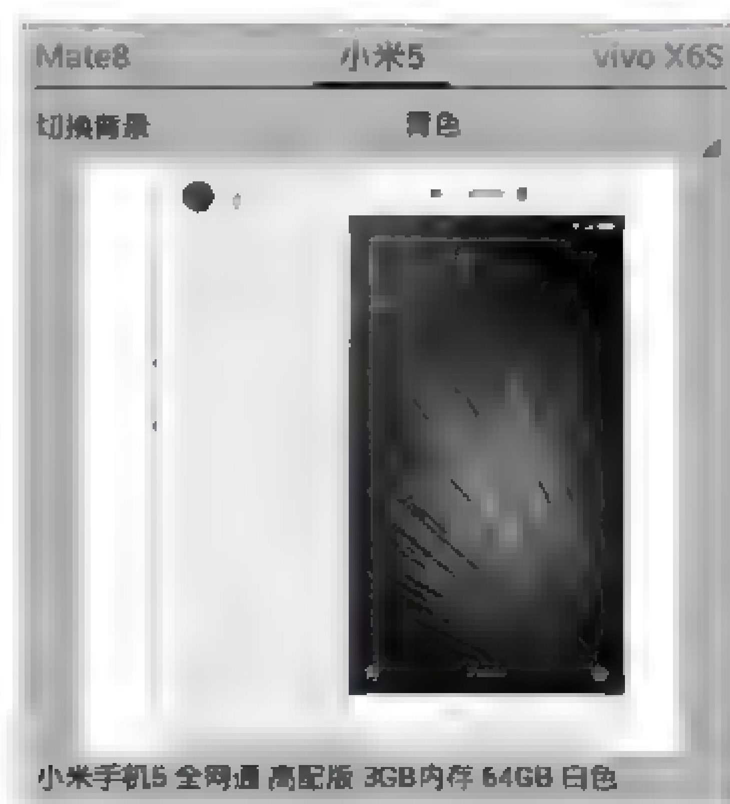


图 5-27　Fragment 发送广播，Activity 接收广播

5.5.2 定时器 AlarmManager

AlarmManager 是 Android 提供的一个全局定时器，利用系统闹钟定时发送广播。这样做的好处是：如果 App 提前注册闹钟的广播接收器，即使 App 退出了，只要定时到达，App 就会被唤醒响应广播事件。是不是很神奇？App 都不在了还能自动响应？没错，就是这样，要不然 Broadcast 怎么对得起大法的名号。

下面来看这种奇妙的事情是如何实现的。首先在页面代码中通过 AlarmManager 设置闹钟，具体代码如下：

```
    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_alarm) {
            Intent intent = new Intent(ALARM_EVENT);
            PendingIntent pIntent = PendingIntent.getBroadcast(this, 0, intent,
                    PendingIntent.FLAG_UPDATE_CURRENT);
            AlarmManager alarmMgr = (AlarmManager) getSystemService(ALARM_SERVICE);
            Calendar calendar = Calendar.getInstance();
            calendar.setTimeInMillis(System.currentTimeMillis());
            calendar.add(Calendar.SECOND, mDelay);
            alarmMgr.set(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), pIntent);
            mDesc = DateUtil.getNowTime() + " 设置闹钟";
            tv_alarm.setText(mDesc);
        }
    }
```

然后在页面代码中定义一个广播接收器 AlarmReceiver，示例代码如下：

```
    private String ALARM_EVENT = "com.example.senior.AlarmActivity.AlarmReceiver";
    private static String mDesc = "";
    private static boolean bArrived = false;
    public static class AlarmReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            if (intent != null) {
                Log.d(TAG, "AlarmReceiver onReceive");
                if (tv_alarm != null && bArrived == false) {
                    bArrived = true;
                    mDesc = String.format("%s\n%s 闹钟时间到达", mDesc,
DateUtil.getNowTime());
                    tv_alarm.setText(mDesc);
                }
            }
        }
    }
```



接着打开 AndroidManifest.xml，在 application 节点下增加广播接收器的声明（注意，凡是在 AndroidManifest.xml 中声明的，就叫静态注册；在代码中声明叫动态注册）：

```
<receiver android:name=".AlarmActivity$AlarmReceiver" >
    <intent-filter>
        <action android:name="com.example.senior.AlarmActivity.AlarmReceiver" />
    </intent-filter>
</receiver>
```

最后的演示界面如图 5-28 和图 5-29 所示。其中，图 5-28 是开始设置闹钟时的界面，图 5-29 是收到闹钟广播时的界面。



图 5-28　开始设置闹钟



图 5-29　收到闹钟广播

这里的定时器能够完美实现广播功能，就是 AlarmManager 与 PendingIntent 相互配合的成果。

PendingIntent 的意思是延迟的意图，只要不是立即传递的消息，都要用 PendingIntent。与之相对的，平常我们使用 Activity 与 Broadcast 传递消息都要求立即处理，所以用 Intent。闹钟有延迟，所以必须用 PendingIntent，PendingIntent 调用了 getBroadcast 方法，表示这次携带的消息用于发送广播。

AlarmManager 的 set 方法用于设置一次性定时器，方法的第一个参数表示定时器类型（一般是 AlarmManager.RTC_WAKEUP，表示定时器即使在睡眠状态下也会启用），第二个参数表示任务的执行时间，第三个参数表示携带消息的延迟任务（getBroadcast 返回的 PendingIntent 对象）。

5.6　实战项目：日历/日程表

本章介绍了好几个高级控件，如此一来，实战项目的功能也变得较为复杂了。本节的实战项目“日历/日程表”包含两个实战项目，一个日历，一个日程表。通过实战项目的练习可以更好地掌握高级控件的用法。

5.6.1　设计思路

手机上的日历一般是一个月一个页面，一年 12 个月就是 12 个页面。日历展示的信息有公历日、农历日，还有常见节假日和二十四节气。相信大家对日历都很熟悉，这里不再赘述。图 5-30 所示为 2016 年 10 月份的日历页，图 5-31 所示为 2016 年 12 月份的日历页。

图 5-30　2016 年 10 月的日历页

图 5-31　2016 年 12 月的日历页

首先数一数日历项目用到了本章的哪些新知识，只看效果图大概会发现以下 6 个。

（1）网格视图 GridView：每月的日期项采用了 GridView，每行 7 列。

（2）基本适配器 BaseAdapter：网格项要展示公历日、农历日、节日与节气，需要适配器配合。

（3）翻页视图 ViewPager：一年 12 个月，支持左右滑动，用到了 ViewPager。

（4）碎片 Fragment：12 个月对应 12 个页面，每个页面都是一个 Fragment。

（5）碎片适配器 FragmentStatePagerAdapter：把 12 个 Fragment 组装到 ViewPager 中。

（6）选项卡标题栏 PagerTabStrip：日历上方每个月的月份标题，对应 PagerTabStrip。

这样看来，日历项目覆盖的知识点有点少，要想全面、深入地复习本章的大部分知识点，还要重新设计一个日程表项目。日程表不但支持基本的日历信息展示，而且支持用户设定每天的日程安排，还支持日程提醒时间。如此一来，日程表项目分成两个页面，一个是类似日历的主页面，另一个是查看日程详情的页面。图 5-32 所示为日程表的主页面，因为要展示每日的日程摘要，所以每天占用一行、一个页面展示七行（一周的日历）。点击每行日历进入日程安排页面，如图 5-33 所示。当天无安排就新增日程，已有安排就查看日程详情。

图 5-32　日程表主页面

图 5-33　日程表详情页

有了效果图，再来看日程表项目用到的知识点，仔细找找你会发现几个？

- 列表视图 ListView：每页日历包含 7 天（一周的日期），采用了 ListView。
- 基本适配器 BaseAdapter：列表项要展示当天的公历日、农历日、节日与节气，还要展示当天的日程安排标题，需要适配器配合。
- 翻页视图 ViewPager：每页一周，一年 52 周，支持左右滑动，用到了 ViewPager。
- 碎片 Fragment：52 周对应 52 个页面，每个页面都是一个 Fragment。
- 碎片适配器 FragmentStatePagerAdapter：把 52 个 Fragment 组装到 ViewPager 中。
- 选项卡标题栏 PagerTabStrip：日历上方每周的周数标题对应 PagerTabStrip。
- 广播 Broadcast：每页根据当周的节日设置背景图，如国庆节所在周显示华表背景，中秋节所在周显示圆月背景，当周无节日显示晴天背景。由 Fragment 通知 Activity 变更背景，上一节说过可用广播技术，Fragment 发送广播，Activity 接收广播。
- 时间选择对话框 TimePickerDialog：设置日程安排要选择日程时间，即时间选择对话框。
- 定时器 AlarmManager：设置日程提醒时间，一般要指定提前若干分钟，这个定时任务就靠 AlarmManager。

另外，还包括其他已经学过的控件知识，如 TextView、Button、EditText、Spinner、SQLite 等，没法一一列举，有待读者在实战中继续巩固提高。

5.6.2 小知识：震动器 Vibrator

上面日程提醒可采用手机震动的方式，会用到震动器 Vibrator，对象从系统服务 VIBRATOR_SERVICE 中获取。震动器的主要方法如下：

- hasVibrator：判断设备是否拥有震动器。
- vibrate：震动手机。可设定单次震动的时长、多次震动的时长、是否重复震动等。
- cancel：取消震动。

使用震动器要在 AndroidManifest.xml 中加上如下权限：

```
<!-- 震动 -->
<uses-permission android:name="android.permission.VIBRATE" />
```

控制手机震动的代码很简单，下面短短两行就实现了震动功能。

```
Vibrator vibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
vibrator.vibrate(3000);   //持续震动 3 秒
```

5.6.3 代码示例

本章的实战项目采用了大量适配器与碎片，此时不仅需要考虑具体编码，还得考虑代码的架构。因为适配器和碎片都分布在单独的代码文件中，所以有必要用另外的 package 包管理，这样不会跟 Activity 文件混在一起。

于是，接下来的编码过程多出了一步，共分为 5 步。

步骤 01 设计代码架构。初步拆分后的 package 包分为以下 7 部分。

- com.example.schedule.activity：存放 Acitivity 页面的代码。
- com.example.schedule.adapter：存放适配器的代码，包括基本适配器和碎片适配器。
- com.example.schedule.bean：存放实体数据结构的代码，如日程表的字段信息。
- com.example.schedule.database：存放读写 SQLite 的数据库操作代码。
- com.example.schedule.fragment：存放碎片代码。
- com.example.schedule.receiver：存放广播接收器的代码。
- com.example.schedule.util：存放工具类的代码。


注 意

本章的演示工程因为加入了许多示例代码，所以包名与相关结构与上述 package 架构不尽相同，这是难以避免的。实战项目作为一个独立的 App，不应混入其他无关代码，建议读者自己开发时按照更清晰的 package 架构编码。

02 想好代码文件与布局文件的名称。比如日历页面的代码文件取名 CalendarActivity.java，对应的布局文件是 activity_calendar.xml；日程表页面的代码文件取名 ScheduleActivity.java，对应的布局文件是 activity_schedule.xml；日程详情页面的代码文件取名 ScheduleDetailActivity.java，对应的布局文件是 activity_schedule_detail.xml；另外，还有一个全局应用的代码文件 MainApplication.java。不要忘了闹钟广播接收器的代码文件 AlarmReceiver.java，还有适配器与碎片的代码及其布局文件，读者可自行构思。

03 在 AndroidManifest.xml 中补充相应配置。在 AndroidManifest.xml 中补充相应配置，主要有以下 3 点。

（1）注册 3 个页面的 acitivity 节点，注册代码如下：

```
<activity android:name=".activity.CalendarActivity" />
<activity android:name=".activity.ScheduleActivity" />
<activity android:name=".activity.ScheduleDetailActivity" />
```

（2）注册闹钟接收器的 receiver 节点，注册代码如下：

```
<receiver android:name=".receiver.AlarmReceiver" >
    <intent-filter>
        <action android:name="com.example.senior.ScheduleDetailActivity.AlarmReceiver" />
    </intent-filter>
</receiver>
```

（3）声明手机震动的操作权限。

04 在 res/drawable 目录加入日程表用到的背景图，在 res/layout 目录下编写布局文件。

05 进行 java 代码开发，包括页面、适配器、碎片、广播接收器等编码。与日历有关的公历计算、农历计算、二十四节气计算都有相应的开源代码，这里只需完成控件操作代码。

编码完成后，日程表主页面应该能够展示每日的日程安排文字，如图 5-34 所示。

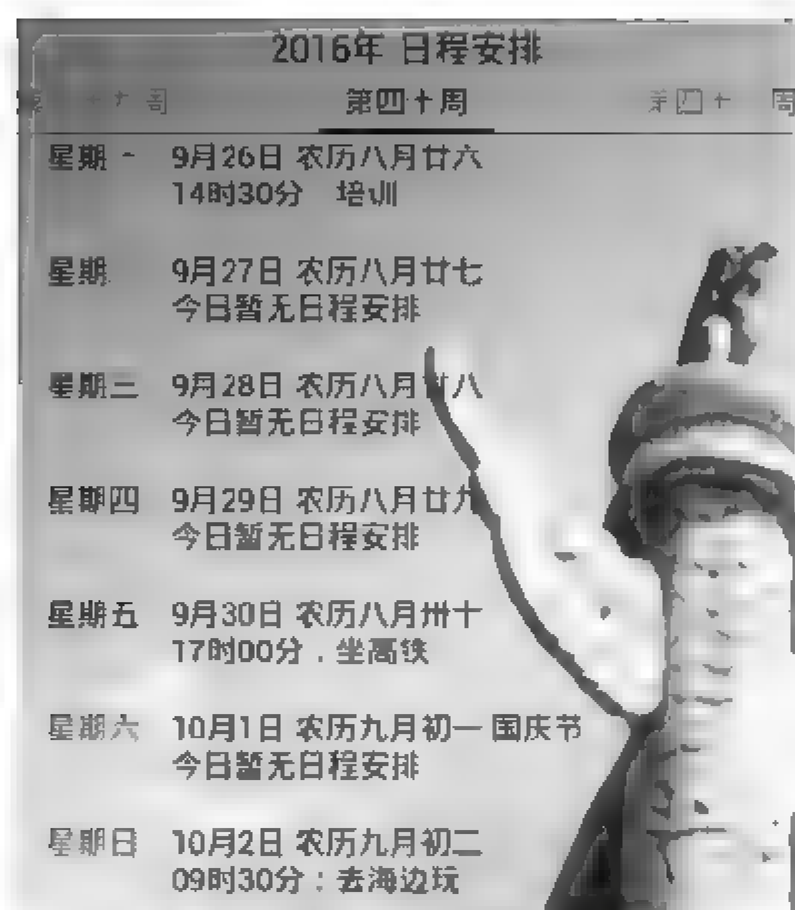


图 5-34 日程表主页面显示每日的日程安排

下面是日程详情页面 ScheduleDetailActivity.java 的主要代码片段：

```
private String[] alarmArray = {"不提醒", "提前 5 分钟", "提前 10 分钟",
        "提前 15 分钟", "提前半小时", "提前 1 小时", "当前时间后 10 秒"};
private int[] advanceArray = {0, 5, 10, 15, 30, 60, 10};
private int alarmType = 0;
class AlarmSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        alarmType = arg2;
    }

    public void onNothingSelected(AdapterView<?> arg0) {
    }
}

@Override
protected void onResume() {
    super.onResume();
    arrange = new ScheduleArrange();
    arrangeHelper = new ScheduleArrangeHelper(this, DbHelper.db_name, null, 1);
    List<ScheduleArrange> arrangeList = (List<ScheduleArrange>)
arrangeHelper.queryInfoByDay(day);
    if (arrangeList.size() >= 1) {
        enableEdit(false);
        arrange = arrangeList.get(0);
        schedule_time.setText(arrange.hour+":"+arrange.minute);
        schedule_alarm.setSelection(arrange.alarm_type);
        schedule_title.setText(arrange.title);
        schedule_content.setText(arrange.content);
```

```
            } else {
                enableEdit(true);
            }
        }

        @Override
        protected void onPause() {
            super.onPause();
            arrangeHelper.close();
        }

        @Override
        public void onClick(View v) {
            int resid = v.getId();
            if (resid == R.id.schedule_time) {
                Calendar calendar = Calendar.getInstance();
                TimePickerDialog dialog = new TimePickerDialog(this, this,
                        calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE),
true);
                dialog.show();
            } else if (resid == R.id.btn_back) {
                finish();
            } else if (resid == R.id.btn_edit) {
                enableEdit(true);
            } else if (resid == R.id.btn_save) {
                if (schedule_title.getText().toString().equals("") == true) {
                    Toast.makeText(this, "请输入日程标题", Toast.LENGTH_SHORT).show();
                    return;
                }
                enableEdit(false);
                String[] time_split = schedule_time.getText().toString().split(":");
                arrange.hour = time_split[0];
                arrange.minute = time_split[1];
                arrange.alarm_type = alarmType;
                arrange.title = schedule_title.getText().toString();
                arrange.content = schedule_content.getText().toString();
                if (arrange.xuhao <= 0) {
                    arrange.month = month;
                    arrange.day = day;
                    arrangeHelper.add(arrange);
                } else {
                    arrangeHelper.update(arrange);
                }
```

```
                Toast.makeText(this, "保存日程成功", Toast.LENGTH_SHORT).show();
                //设置提醒闹钟
                if (alarmType > 0) {
                    Intent intent = new Intent(ALARM_EVENT);
                    PendingIntent pIntent = PendingIntent.getBroadcast(this, 0, intent,
                            PendingIntent.FLAG_UPDATE_CURRENT);
                    AlarmManager alarmMgr = (AlarmManager) getSystemService(ALARM_SERVICE);
                    Calendar calendar = Calendar.getInstance();
                    if (alarmType == 6) {
                        calendar.setTimeInMillis(System.currentTimeMillis());
                        calendar.add(Calendar.SECOND, advanceArray[alarmType]);
                    } else {
                        int day_int = Integer.parseInt(day);
                        calendar.set(day_int/10000, day_int%10000/100-1, day_int%100,
                                Integer.parseInt(arrange.hour), Integer.parseInt(arrange.minute), 0);
                        calendar.add(Calendar.SECOND, -advanceArray[alarmType]*60);
                    }
                    alarmMgr.set(AlarmManager.RTC_WAKEUP, calendar.getTimeInMillis(), pIntent);
                }
            }
        }

        @Override
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            String time = String.format("%02d:%02d", hourOfDay, minute);
            schedule_time.setText(time);
        }

        private String ALARM_EVENT = "com.example.senior.ScheduleDetailActivity.AlarmReceiver";
        public static class AlarmReceiver extends BroadcastReceiver {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent != null) {
                    Log.d(TAG, "AlarmReceiver onReceive");
                    Vibrator vibrator = (Vibrator)
context.getSystemService(Context.VIBRATOR_SERVICE);
                    vibrator.vibrate(3000); // 默认震动3秒
                }
            }
        }
```

5.7 小　　结

本章主要介绍了 App 开发的高级控件相关知识，包括日期时间控件的用法（日期选择器、时间选择器）、列表类视图的用法（基本适配器、列表视图、网格视图）、翻页类视图的基本用法（翻页视图、翻页适配器、翻页标题栏）、碎片的用法（静态注册方式、动态注册方式、碎片适配器）、Broadcast 组件的基本用法（发送广播、接收、定时器广播）。中间穿插了实战模块的运用，如改进后的购物车、改进后的启动引导页等。最后设计了一个实战项目“日历/日程表”，在该项目的 App 编码中，采用本章介绍的大部分控件与适配器，以及广播发送和广播接收器的处理。另外，介绍了震动器的用法。

通过本章的学习，读者应该能够掌握以下 3 种开发技能：

（1）在布局文件中合理使用本章学到的控件。

（2）在代码中合理调用本章学到的控件和适配器的相关方法。

（3）学会 Broadcast 组件的用法，如在不同组件之间发送与接收广播、设置定时器并接收定时器广播等。



自定义控件

本章介绍 App 开发经常涉及的自定义控件相关技术，主要包括自定义视图的过程与步骤、自定义动画的原理与实现、自定义对话框的概念与示例、自定义通知栏的用法与定制，另外介绍四大组件之一的 Service 的生命周期与启停方式。最后结合本章所学的知识，演示一个实战项目“手机安全助手”的设计与实现。

6.1　自定义视图

本节介绍自定义视图的过程，包括声明属性与编写代码两个过程。编写代码的过程分为构造对象、测量尺寸、绘制视图 3 个步骤。另外，详细说明绘制视图的 3 种途径：重写 onLayout 方法、重写 onDraw 方法和重写 dispatchDraw 方法。

6.1.1　声明属性

Android 自带的视图有时无法满足实际需求，这种情况下开发者就得自定义视图。自定义视图好比自己造车，造车总比开车难很多，不过只要找到窍门，其实也没有想象得那么难。自定义视图涉及许多概念，为了使读者更容易理解，下面从一个小例子入手，先产生感性认识再学习理论知识。

第 5 章提到 PagerTitleStrip 和 PagerTabStrip 无法在布局文件中指定文字样式，只能在代码中设置，让人很不习惯，如果可以直接指定 textColor 和 textSize 属性就会好很多。现在我们小试牛刀，扩展自定义属性，以满足在布局文件指定属性的要求。具体步骤如下：

步骤01 在 res\values 目录下创建 attrs.xml。其中，declare-styleable 的 name 属性值表示新视图名为 PagerTab，两个 attr 节点表示新增的两个属性分别是 textColor 和 textSize。文件内容如下：

```
<resources>
    <declare-styleable name="PagerTab">
        <attr name="textColor" format="color" />
        <attr name="textSize" format="dimension" />
    </declare-styleable>
</resources>
```

步骤02 在代码目录中创建 PagerTab.java，填入以下代码：

```
public class PagerTab extends PagerTabStrip {
    private int textColor = Color.BLACK;
    private int textSize = 15;

    public PagerTab(Context context) {
        super(context);
    }

    public PagerTab(Context context,AttributeSet attrs) {
        super(context, attrs);
        if (attrs != null) {
        TypedArray attrArray=getContext().obtainStyledAttributes(attrs, R.styleable.PagerTab);
        textColor = attrArray.getColor(R.styleable.PagerTab_textColor, textColor);
        textSize = attrArray.getDimensionPixelSize(R.styleable.PagerTab_textSize, textSize);
```

```
            attrArray.recycle();
        }
    }

    @Override
    protected void onDraw(Canvas canvas) {
        setTextColor(textColor);
        setTextSize(TypedValue.COMPLEX_UNIT_SP, textSize);
        super.onDraw(canvas);
    }
}
```

03 在布局文件根节点增加命名空间声明 xmlns:app="http://schemas.android.com/apk/res-auto"，并把 android.support.v4.view.PagerTabStrip 的节点名称改为自定义视图的全路径名称（如 com.example.custom.widget.PagerTab），同时在该节点下指定新增的两个属性——app:textColoe 与 app:textSize。修改后的布局文件代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <android.support.v4.view.ViewPager
        android:id="@+id/vp_content"
        android:layout_width="match_parent"
        android:layout_height="400dp" >

        <com.example.custom.widget.PagerTab
            android:id="@+id/pts_tab"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            app:textColor="@color/red"
            app:textSize="17sp" />
    </android.support.v4.view.ViewPager>
</LinearLayout>
```

完成以上代码的修改后运行 App，实现的效果如图 6-1 所示，此时翻页标题栏的文字颜色变为红色，字体也变大了。

在自定义视图的步骤 1 中，attr 节点的 name 表示新属性的名称，format 表示新属性的格式（数据类型）；在步骤 2 中，调用 getColor 方法获取颜色值，调用 getDimensionPixelSize 方法获取文字大小，不同的数据类型调用不同的获取方法。有关属性类型及其获取方法的对应说明见表 6-1。



图 6-1　自定义的翻页标题栏

表6-1　属性类型的取值说明

属性类型	获取方法	说明
boolean	getBoolean	布尔值。取值为 true 或 false
integer	getInt	整型值
float	getFloat	浮点值
string	getString	字符串
color	getColor	颜色值。取值为开头带#的六位或八位十六进制数
dimension	getDimension	尺寸值。取值为末尾带 dp 的尺寸数值
dimension	getDimensionPixelSize	字体大小。取值为末尾带 sp 的尺寸数值
fraction	getFraction	百分数。取值为末尾带%的百分数
reference	getResourceId	参考某一资源。取值如@drawable/ic_launcher
enum	getInt	枚举值
flag	getInt	标志位

表 6-1 的 enum 类型与 flag 类型的使用稍微复杂，枚举类型的属性常见的有 LinearLayout 的 orientation 和 ImageView 的 scaleType；标志类型的属性常见的有 TextView 的 gravity 和 EditText 的 inputType。下面是枚举类型的属性声明：

```
<declare-styleable name="PagerTab">
    <attr name="customOrientation">
        <enum name="horizontal" value="0" />
        <enum name="vertical" value="1" />
    </attr>
</declare-styleable>
```

下面是标志类型的属性声明：

```
<declare-styleable name="PagerTab">
    <attr name="customGravity">
        <flag name = "center" value = "0" />
        <flag name = "left" value = "1" />
```

```
            <flag name = "top" value = "2" />
            <flag name = "right" value = "4" />
            <flag name = "bttom" value = "8" />
        </attr>
</declare-styleable>
```

6.1.2 构造对象

新增视图属性的声明很简单，麻烦的是在代码中进行视图的自定义处理。自定义视图的编码主要由3部分组成：

（1）重写构造函数，初始化该视图的自有属性。

（2）重写测量函数 onMesure，计算该视图的宽与高（一般只有复杂视图才重写该函数）。

（3）重写绘图函数 onLayout、onDraw、dispatchDraw，视情况重写3个中的一个或多个。

一般要重写3个构造函数。前面在演示新控件 PagerTab 时，示例代码给出了3个构造函数（实际只实现了两个），分别是：

（1）只带一个参数的 public PagerTab(Context context)。在代码中声明对象时采用该构造函数。

（2）带两个参数的 public PagerTab(Context context,AttributeSet attrs)。在布局文件中引用自定义视图时采用该构造函数。

（3）带3个参数的 public PagerTab(Context context, AttributeSet attrs, int defStyleAttr)。该构造函数的作用是：除了布局文件中指定的属性，另外在代码中指定默认风格。第3个参数 defStyleAttr 是一种特殊属性，类型既非整型也非字符串，而是参照类型（reference，需要在 styles.xml 中另外定义）。具体使用步骤如下：

步骤 01 在 styles.xml 中定义一种风格样式。

步骤 02 在 attrs.xml 中声明该风格样式的参照属性，举例如下：

```
<attr name="CustomizeStyle" format="reference" />
```

步骤 03 在代码中由第二种构造函数调用第三种构造函数，在调用时把该参照属性传到第三个参数中，示例代码如下：

```
public PagerTab(Context context, AttributeSet attrs) {
      this(context, attrs, R.attr.CustomizeStyle);
}

public PagerTab(Context context, AttributeSet attrs, int defStyleAttr) {
      super(context, attrs, defStyleAttr);
            if (attrs != null) {
            TypedArray attrArray = getContext().obtainStyledAttributes( attrs, R.styleable.PagerTab,
defStyleAttr, R.style.DefaultCustomizeStyle);
```

```
                //此处省略各个属性值的读取
                attrArray.recycle();
            }
        }
```

这样，系统在寻找该视图的属性时就会先找布局文件，再找 attrs.xml 中声明的 R.attr.CustomizeStyle 风格样式，最后找 styles.xml 中 R.style.DefaultCustomizeStyle 的风格样式。第三种构造函数用的不多，无须深入研究，了解即可。

6.1.3 测量尺寸

自定义视图的第二步是测量尺寸。大家知道，添加视图的目的是在屏幕上显示期望的图案。如此，在绘制图案之前系统得先知道这个图案的尺寸，即宽和高。一般在布局文件中对视图的宽和高有 3 种赋值方式，具体说明见表 6-2。

表6-2 尺寸大小的3种赋值方式

XML 中的尺寸类型	LayoutParams 类的尺寸类型	说明
match_parent	MATCH_PARENT	与上级视图大小一样
wrap_content	WRAP_CONTENT	按照自身尺寸进行适配
**dp	整型数	具体的尺寸数值

方式 1 和方式 3 都好办，要么取上级视图的数值，要么取具体数值。难办的是方式 2，这个尺寸究竟要如何度量，不可能让开发者人手一把尺子在屏幕上比划。Android 提供了相关度量方法，可以在不同情况下进行尺寸测量。需要测量的对象主要有 3 种，分别是文本尺寸、图形尺寸和布局尺寸。

1. 文本尺寸测量

文本尺寸分为文本的宽度和高度，要根据文本大小分别进行计算。其中，文本宽度使用 Paint 类的 measureText 方法测量，具体代码如下：

```
    public static float getTextWidth(String text, float textSize) {
        if (text==null || text.length()<=0) {
            return 0;
        }
        Paint paint = new Paint();
        paint.setTextSize(textSize);
        return paint.measureText(text);
    }
```

文本高度的计算要烦琐一些，用到了 FontMetrics 类，该类提供了 5 个与高度相关的属性，详细说明见表 6-3。

表6-3 FontMetrics类的距离属性说明

FontMetrics 类的距离属性	说明
top	行的顶部与基线的距离
ascent	字符的顶部与基线的距离
descent	字符的底部与基线的距离
bottom	行的底部与基线的距离
leading	行间距

之所以区分这些属性，是为了计算不同规格的高度。如果要得到文本自身的高度，高度值就是 descent 减去 ascent；如果要得到文本所在行的行高，高度值就是 bottom 减去 top 再加上 leading。具体的高度计算代码如下：

```
public static float getTextHeight(String text, float textSize) {
    Paint paint = new Paint();
    paint.setTextSize(textSize);
    FontMetrics fm = paint.getFontMetrics();
    return fm.descent - fm.ascent;  //文本自身的高度
    //return fm.bottom - fm.top + fm.leading;  //文本所在行的行高
}
```

下面看看文本尺寸的度量结果，当字体大小为 17sp 时，示例文本的宽度为 119、高度为 19，如图 6-2 所示；当字体大小为 25sp 时，示例文本的宽度为 175、高度为 29，如图 6-3 所示。

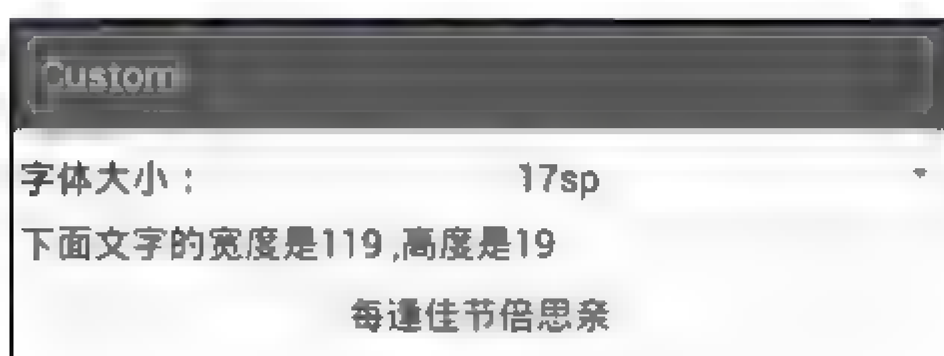


图 6-2 字体大小为 17sp 时的尺寸

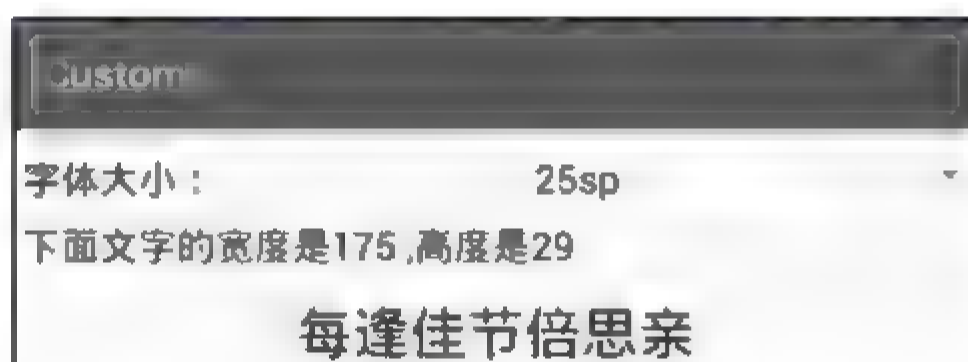


图 6-3 字体大小为 25sp 时的尺寸

2. 图形尺寸测量

相对于文本尺寸，图形尺寸的计算反而简单些，因为 Android 提供了可以直接使用的宽、高获取方法。如果图形是 Bitmap 格式，就通过 getWidth 和 getHeight 方法获取位图对象的宽度和高度；如果图形是 Drawable 格式，就通过 getIntrinsicWidth 方法获取该图形的宽度，通过 getIntrinsicHeight 方法获取该图形的高度。

3. 布局尺寸测量

文本尺寸测量主要用于 TextView、Button 等文本控件，图形尺寸测量主要用于 ImageView、ImageButton 等图像控件。在实际开发中，有更多场合需要测量布局视图的尺寸。布局视图内部可能有文本控件、图像控件，还可能有 padding 和 margin。如此一来，对布局视图的内部控件一个个单独测量变得不切实际。View 类提供了一种对布局整体进行测量的思路。对应

layout_width 和 layout_height 的 3 种赋值方式，Android 的视图提供了 3 种测量模式，具体取值说明见表 6-4。

表6-4　测量模式的取值说明

MeasureSpec 类的测量模式	视图宽、高的赋值方式	说明
AT_MOST	MATCH_PARENT	达到最大
UNSPECIFIED	WRAP_CONTENT	未指定（实际就是自适应）
EXACTLY	具体 dp 值	精确尺寸

围绕这 3 种模式衍生了相关度量方法，如 ViewGroup 类的 getChildMeasureSpec 方法、MeasureSpec 类的 makeMeasureSpec 方法、View 类的 measure 方法等。具体的测量原理可以不用深究，下面直接切入正题，看下测量尺寸的实现代码：

```
    public static float getRealHeight(View child) {
        LinearLayout llayout = (LinearLayout) child;
        ViewGroup.LayoutParams params = llayout.getLayoutParams();
        if (params == null) {
            params = new ViewGroup.LayoutParams(
                    LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
        }
        int widthSpec = ViewGroup.getChildMeasureSpec(0, 0, params.width);
        int heightSpec;
        if (params.height > 0) {
            heightSpec = View.MeasureSpec.makeMeasureSpec(params.height,
MeasureSpec.EXACTLY);
        } else {
            heightSpec = View.MeasureSpec.makeMeasureSpec(0, MeasureSpec.UNSPECIFIED);
        }
        llayout.measure(widthSpec, heightSpec);
        return llayout.getMeasuredHeight();   // 获得布局高度，获得布局宽度可调用
getMeasuredWidth
    }
```

现在很多 App 页面都提供下拉刷新功能，需要计算下拉刷新的头部高度，以便在下拉时判断整个页面要拉动多少距离。下面演示下拉刷新的头部高度，如图 6-4 所示。头部布局中有图像、文字和间隔，调用 getRealHeight 方法计算得到的布局高度为 170。



图 6-4　布局视图的高度测量结果

6.1.4　绘制视图

在自定义视图中，可重写 3 个函数用于视图的绘制，分别是 onLayout、onDraw 和 dispatchDraw。这 3 个函数的执行顺序是 onLayout→onDraw→dispatchDraw。其中，onLayout 和 dispatchDraw 通常用于布局类视图。下面逐一介绍这 3 个函数的用途与用法。

1. onLayout

onLayout 方法用于定位子视图在本布局视图中的位置。该方法的入参表示本布局在上级视图的上、下、左、右位置，子视图在本布局中的位置要另行计算，计算完毕调用子视图的 layout 方法调整子视图的位置。

为直观理解 onLayout 的用法，下面给出自定义偏移布局的代码：

```
public class OffsetLayout extends AbsoluteLayout {
    private int mOffsetHorizontal = 0, mOffsetVertical = 0;
    public OffsetLayout(Context context) {
        super(context);
    }

    public OffsetLayout(Context context,AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onLayout(boolean changed, int l, int t, int r, int b) {
        int count = getChildCount();
        for (int i = 0; i < count; i++) {
            View child = getChildAt(i);
            if (child.getVisibility() != GONE) {
                int new_left = (r-l)/2-child.getMeasuredWidth()/2+mOffsetHorizontal;
                int new_top = (b-t)/2-child.getMeasuredHeight()/2+mOffsetVertical;
                    child.layout(new_left, new_top,
                            new_left+child.getMeasuredWidth(), new_top+child.getMeasuredHeight());
            }
        }
    }

    public void setOffsetHorizontal(int offset) {
        mOffsetHorizontal = offset;
        mOffsetVertical = 0;
        requestLayout();
    }

    public void setOffsetVertical(int offset) {
```

```
            mOffsetHorizontal = 0;
            mOffsetVertical = offset;
            requestLayout();
        }
    }
```

该偏移布局可根据设定的偏移值动态调整子视图的偏移位置。页面代码可直接调用 OffsetLayout 对象的 setOffsetHorizontal 方法或 setOffsetVertical 方法，完成水平或垂直方向的偏移值设置。如图 6-5～图 6-8 所示，这 4 张图分别展示了不同偏移值的效果。其中，图 6-5 所示为无偏移，图 6-6 所示为向左偏移 100，图 6-7 所示为向右偏移 100，图 6-8 所示为向上偏移 100。



图 6-5　无偏移的情况



图 6-6　向左偏移 100



图 6-7　向右偏移 100



图 6-8　向上偏移 100

2. onDraw

onDraw 是最常使用的绘图方法，该方法的入参为 Canvas 画布对象，在画布上绘图相当于在屏幕上绘图。绘图本身是个很大的课题，画布的用法也多种多样，如 Canvas 提供了 3 类方法，分别是划定可绘制的区域、在区域内部绘制图形和画布的控制操作。

（1）划定可绘制的区域

虽然本视图内的所有区域都是可以绘制的，但是有时候我们只想在某个矩形区域内部画画，这时在绘图之前就得指定允许绘图的区域界限，相关方法说明如下：

- clipPath：裁剪不规则曲线区域。
- clipRect：裁剪矩形区域。
- clipRegion：裁剪一块组合区域。

（2）在区域内部绘制图形

该类方法用来绘制各种基本几何图形，相关方法说明如下：

- drawArc：绘制扇形/弧形。第4个参数为true时画扇形、为false时画弧形。
- drawBitmap：绘制图像。
- drawCircle：绘制圆形。
- drawLine：绘制直线。
- drawOval：绘制椭圆。
- drawPath：绘制路径，即不规则曲线。
- drawPoint：绘制点。
- drawRect：绘制矩形。
- drawRoundRect：绘制圆角矩形。
- drawText：绘制文本。

（3）画布的控制操作

控制操作包括旋转、缩放、平移以及存取画布状态的操作，相关方法说明如下：

- rotate：旋转画布。
- scale：缩放画布。
- translate：平移画布。
- save：保存画布状态。
- restore：恢复画布状态。

上面绘图用的draw***方法只是指定绘制哪个几何图形，真正的细节描绘还要靠画笔Paint类实现。Paint类定义了画笔的颜色、样式、粗细、阴影等，常用方法说明如下：

- setAntiAlias：设置是否使用抗锯齿功能。主要用于画圆圈等曲线。
- setDither：设置是否使用防抖动功能。
- setColor：设置画笔的颜色。
- setShadowLayer：设置画笔的阴影区域与颜色。
- setStyle：设置画笔的样式。Style.STROKE表示线条，Style.FILL表示填充。
- setStrokeWidth：设置画笔线条的宽度。

下面演示不同图形的绘制效果。调用drawRoundRect方法绘制圆角矩形，如图6-9所示。调用drawOval方法绘制椭圆，如图6-10所示。



图 6-9　绘制圆角矩形



图 6-10　绘制椭圆

3. dispatchDraw

dispatchDraw 与 onDraw 函数一样都是绘图方法，区别在于 onDraw 的调用在绘制子视图之前，dispatchDraw 的调用在绘制子视图之后。如果不想自身视图被子视图覆盖，就只能在 dispatchDraw 方法中进行绘图处理。

下面演示如何通过 dispatchDraw 方法进行绘图：

```
public class AfterRelativeLayout extends RelativeLayout {
    private int mDrawType = 0;
    private Paint mPaint = new Paint();

    public AfterRelativeLayout(Context context) {
        this(context, null);
    }

    public AfterRelativeLayout(Context context,AttributeSet attrs) {
        super(context, attrs);
        mPaint.setAntiAlias(true);           //设置画笔为无锯齿
        mPaint.setDither(true);              //防抖动
        mPaint.setColor(Color.BLACK);        //设置画笔颜色
        mPaint.setStrokeWidth(3);            //线宽
        mPaint.setStyle(Style.STROKE);       //画笔类型。STROKE：空心，FILL：实心
    }

    @Override
    protected void dispatchDraw(Canvas canvas) {
        super.dispatchDraw(canvas);
        int width = getMeasuredWidth();
        int height = getMeasuredHeight();
        if (width>0 && height>0) {
            if (mDrawType == 1) {   // 画矩形
```

```
                Rect rect = new Rect(0, 0, width, height);
                canvas.drawRect(rect, mPaint);
            } else if (mDrawType == 2) {  // 画圆角矩形
                RectF rectF = new RectF(0, 0, width, height);
                canvas.drawRoundRect(rectF, 30, 30, mPaint);
            } else if (mDrawType == 3) {  // 画圆圈
                int radius = Math.min(width, height)/2;
                canvas.drawCircle(width/2, height/2, radius, mPaint);
            } else if (mDrawType == 4) {  // 画椭圆
                RectF oval = new RectF(0, 0, width, height);
                canvas.drawOval(oval, mPaint);
            } else if (mDrawType == 5) {  // 画线段
                Rect rect = new Rect(0, 0, width, height);
                canvas.drawRect(rect, mPaint);
                canvas.drawLine(0, 0, width, height, mPaint);
                canvas.drawLine(0, height, width, 0, mPaint);
            }
        }
    }

    public void setDrawType(int type) {
        setBackgroundColor(Color.WHITE);
        mDrawType = type;
        invalidate();
    }
}
```

观察 onDraw 与 dispatchDraw 两种绘图方式的效果对比，如图 6-11 和图 6-12 所示。其中，图 6-11 是重写 onDraw 方法的效果图，可以看到中间的按钮遮住了交叉线；图 6-12 是重写 dispatchDraw 方法的效果图，可以看到交叉线没被按钮遮住，依然显示在视图中央。

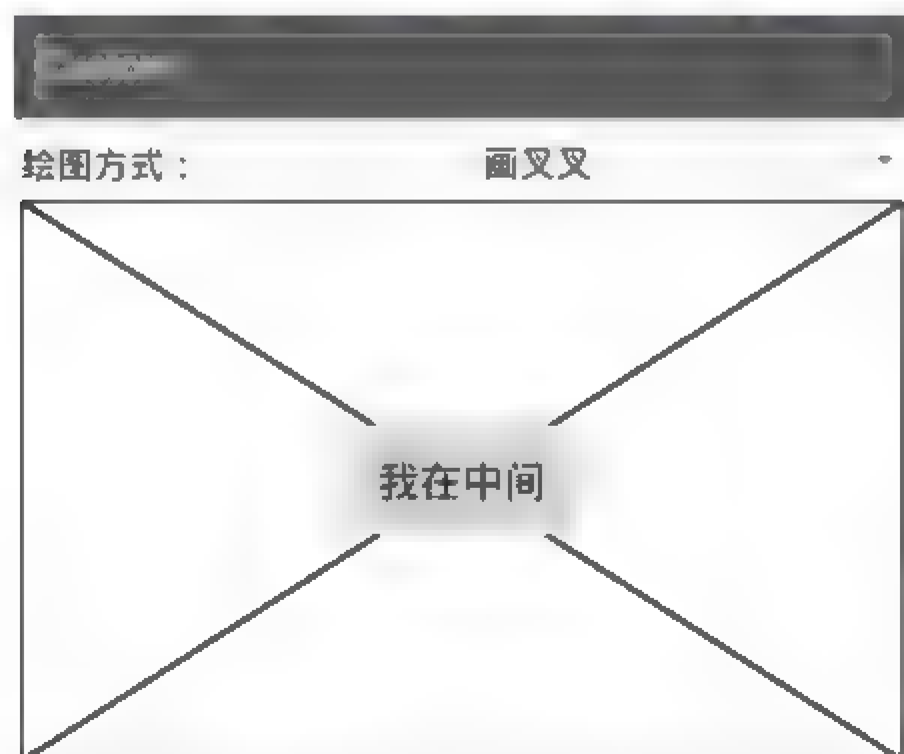


图 6-11　重写 onDraw 方法

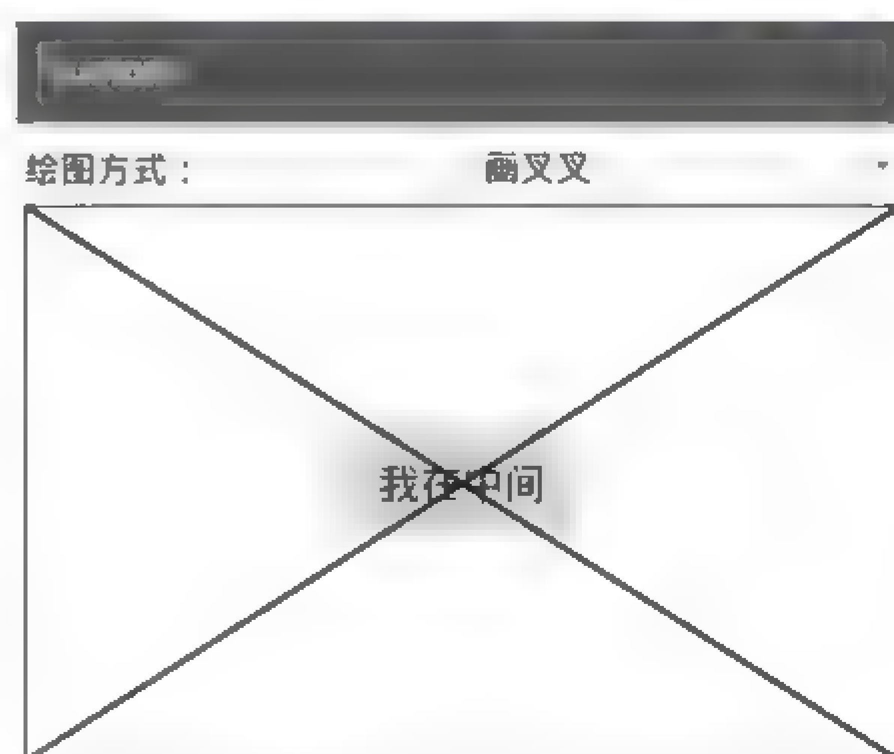


图 6-12　重写 dispatchDraw 方法

总结一下 onLayout、onDraw、dispatchDraw 三个函数的区别：

（1）onLayout 只能调整子视图的位置，而 onDraw 和 dispatchDraw 允许绘制新图形。

（2）onDraw 的调用在绘制子视图之前，而 dispatchDraw 的调用在绘制子视图之后。

（3）onLayout 若想立即显示位置调整后的视图，则要调用 requestLayout 方法；onDraw 和 dispatchDraw 若想立即显示图形绘制后的视图，则要调用 invalidate 方法。

6.2 自定义动画

本节介绍计时器的实现方式和如何利用计时器实现简单的下拉动画，并结合自定义视图的方法实现一个圆弧进度动画的新控件。

6.2.1 任务 Runnable

在前面的章节中，有几个需要延迟处理的地方用到了 Handler+Runnable 组合，即调用 Handler 的 postDelayed 方法延迟若干时间再执行指定的 Runnable 任务。这几处延迟处理主要是为了避免资源冲突，不过延迟处理更多用于动画界面的渲染。

Runnable 接口可声明一连串任务，定义了接下来要做的事情。简单地说，Runnable 接口就是一个代码片段。实现 Runnable 接口只需重写 run 函数，在该方法内部存放要运行的任务代码。run 函数无须显式调用，在启动 Runnable 实例时就会调用对象的 run 方法。

尽管基本视图 View 提供了 post 与 postDelayed 方法用于启动 Runnable 任务，不过实际开发中经常利用 Handler 启动任务。下面是 Handler 处理 Runnable 任务的常见方法说明：

- post：立即启动 Runnable 任务。
- postDelayed：延迟若干时间后启动 Runnable 任务。
- postAtTime：在指定时间启动 Runnable 任务。
- removeCallbacks：移除指定的 Runnable 任务。

计时器是 Runnable 的一个简单应用，与动画的实现原理相关，如电影每秒播放 20 帧画面，连起来就是会动的视频，动画的渲染与之同理。下面是一个简单计时器的代码片段：

```
@Override
public void onClick(View v) {
    if (v.getId() == R.id.btn_runnable) {
        if (bStart == false) {
            btn_runnable.setText("停止计数");
            mHandler.post(mCounter);
        } else {
            btn_runnable.setText("开始计数");
            mHandler.removeCallbacks(mCounter);
        }
        bStart = !bStart;
```

```
        }
    }

    private boolean bStart = false;
    private Handler mHandler = new Handler();
    private int mCount = 0;
    private Runnable mCounter = new Runnable() {
        @Override
        public void run() {
            mCount++;
            tv_result.setText("当前计数值为 ： "+mCount);
            mHandler.postDelayed(this, 1000);
        }
    };
```

计时器的效果如图 6-13 和 6-14 所示。其中，图 6-13 表示当前正在计数；图 6-14 表示当前停止计数，终止的计数值为 21。



图 6-13　计时器开始计数



图 6-14　计时器结束计数

6.2.2　下拉刷新动画

现在我们把计时器引入下拉刷新中，每隔若干时间展示逐步加大的视图偏移，从而实现下拉刷新头部的下拉动画。首先，计算下拉刷新头部的高度，这会用到 6.1 节布局尺寸测量的知识。接着，计时器每隔若干时间为 padding 设置逐步加大的高度偏移。不得不说，padding 大法非常好用，当 padding 为负值时，表示当前视图被遮住了一部分。最后，高度的偏移值达到头部布局的高度时，停止 Runnable 的刷新任务，下拉动画完成。

下面是下拉刷新动画的代码片段：

```
    @Override
    public void onClick(View v) {
        mHeight = (int) MeasureUtil.getRealHeight(ll_header);
        if (v.getId() == R.id.btn_pull) {
            if (bStart == false) {
                mOffset = -mHeight;
                btn_pull.setEnabled(false);
                mHandler.post(mRefresh);
            } else {
                btn_pull.setText("开始刷新");
```

```
                ll_header.setVisibility(View.GONE);
            }
            bStart = !bStart;
        }
    }

    private int mHeight;
    private boolean bStart = false;
    private Handler mHandler = new Handler();
    private int mOffset = 0;
    private Runnable mRefresh = new Runnable() {
        @Override
        public void run() {
            if (mOffset <= 0) {
                ll_header.setPadding(0, mOffset, 0, 0);
                ll_header.setVisibility(View.VISIBLE);
                mOffset += 8;
                mHandler.postDelayed(this, 80);
            } else {
                btn_pull.setText("恢复页面");
                btn_pull.setEnabled(true);
            }
        }
    };
```

下拉刷新动画的效果如图 6-15 和 6-16 所示。其中，图 6-15 展示的是下拉动画进行中的截图；图 6-16 展示的是下拉完毕的截图，此时下拉刷新头部完全显示。



图 6-15　下拉动画开始



图 6-16　下拉动画结束

6.2.3　圆弧进度动画

当用户下载文件或做其他事情时，往往想知道当前到什么进度了。在 Windows 系统中常用细长的进度条表示，在手机上因为屏幕限制，习惯展示圆形或弧形的进度圈。接下来介绍的就是圆弧进度动画，该动画控件正好可以与 6.1 节自定义视图的绘制方法结合起来。既可以复习旧知识，又能巩固新知识。

绘制圆弧动画的主要思路是在一段指定的时间内持续不断地绘制一个扇形或圆弧，连起来整个画面就会动起来。还要进行一些参数设置，如设置该圆圈的位置、开始和结束的角度、



转动的速率，以及画笔的颜色、粗细、样式等。另外，为了区分处理动画的背景和前景，还要分别构造背景视图（用于衬托动画）和前景视图（用于展示圆弧）。

自定义圆弧动画的完整代码如下：

```
public class CircleAnimation extends RelativeLayout {
    private final static String TAG = "CicleAnimation";
    private RectF mRect = new RectF(100, 10, 400, 310);
    private int mBeginAngle = 0, mEndAngle = 270;
    private int mFrontColor = 0xffff0000, mShadeColor = 0xffeeeeee;
    private float mFrontLine = 10, mShadeLine = 10;
    private Style mFrontStyle = Style.STROKE, mShadeStyle = Style.STROKE;
    private ShadeView mShadeView;
    private FrontView mFrontView;
    private int mRate = 2. mDrawTimes = 0, mInterval = 70, mFactor, mSeq = 0, mDrawingAngle = 0;
    private Context mContext;

    public CircleAnimation(Context context) {
        super(context);
        mContext = context;
    }


    public void render() {
        removeAllViews();
        mShadeView = new ShadeView(mContext);
        addView(mShadeView);
        mFrontView = new FrontView(mContext);
        addView(mFrontView);
        play();
    }

    public void play() {
        mSeq = 0;
        mDrawingAngle = 0;
        mDrawTimes = mEndAngle/mRate;
        mFactor = mDrawTimes/mInterval + 1;
        Log.d(TAG, "mDrawTimes="+mDrawTimes+",mInterval="+mInterval+",mFactor="+mFactor);
        mFrontView.invalidateView();
    }

    public void setRect(int left, int top, int right, int bottom) {
        mRect = new RectF(left, top, right, bottom);
    }

    public void setAngle(int begin_angle, int end_angle) {
        mBeginAngle = begin_angle;
        mEndAngle = end_angle;
```

```
	}
	//speed:每次移动几个度数　　frames:每秒移动几帧
	public void setmRate(int speed, int frames) {
		mRate = speed;
		mInterval = 1000/frames;
	}
	public void setFront(int color, float line, Style style) {
		mFrontColor = color;
		mFrontLine = line;
		mFrontStyle = style;
	}
	public void setShade(int color, float line, Style style) {
		mShadeColor = color;
		mShadeLine = line;
		mShadeStyle = style;
	}

	private class ShadeView extends View {
		private Paint paint;
		public ShadeView(Context context) {
			super(context);
			paint = new Paint();
			paint.setAntiAlias(true);
			paint.setDither(true);
			paint.setColor(mShadeColor);
			paint.setStrokeWidth(mShadeLine);
			paint.setStyle(mShadeStyle);
		}
		@Override
		protected void onDraw(Canvas canvas) {
			super.onDraw(canvas);
			canvas.drawArc(mRect, mBeginAngle, 360, false, paint);
		}
	}
	private class FrontView extends View {
		private Paint paint;
		private Handler handler = new Handler();
		public FrontView(Context context) {
			super(context);
			paint = new Paint();
			paint.setAntiAlias(true);			//设置画笔为无锯齿
```

```
            paint.setDither(true);                  //防抖动
            paint.setColor(mFrontColor);            //设置画笔颜色
            paint.setStrokeWidth(mFrontLine);       //线宽
            paint.setStyle(mFrontStyle);            //画笔类型。STROKE：空心，FILL：实心
            //paint.setStrokeJoin(Paint.Join.ROUND);     //画笔接洽点类型。影响矩形直角的外轮廓
            paint.setStrokeCap(Paint.Cap.ROUND);       //画笔笔刷类型。影响画笔的始末端
        }

        @Override
        protected void onDraw(Canvas canvas) {
            super.onDraw(canvas);
            canvas.drawArc(mRect, mBeginAngle, (float) (mDrawingAngle), false, paint);
        }

        public void invalidateView(){
            handler.postDelayed(drawRunnable, 0);
        }

        private Runnable drawRunnable = new Runnable() {
            @Override
            public void run() {
                if (mDrawingAngle >= mEndAngle) {
                    mDrawingAngle = mEndAngle;
                    invalidate();
                    handler.removeCallbacks(drawRunnable);
                } else {
                    mDrawingAngle = mSeq*mRate;
                    mSeq++;
                    handler.postDelayed(drawRunnable, (long) (mInterval-mSeq/mFactor));
                    invalidate();
                }
            }
        };
    }
}
```

要在活动页面中显示圆弧动画，可加入以下代码：

```
        mAnimation = new CircleAnimation(this);
        ll_layout.addView(mAnimation);
        mAnimation.render();
```

圆弧动画的效果如图 6-17 和 6-18 所示。其中，图 6-17 展示的是圆弧动画进行中的截图，图 6-18 展示的圆弧动画播放完成的截图。



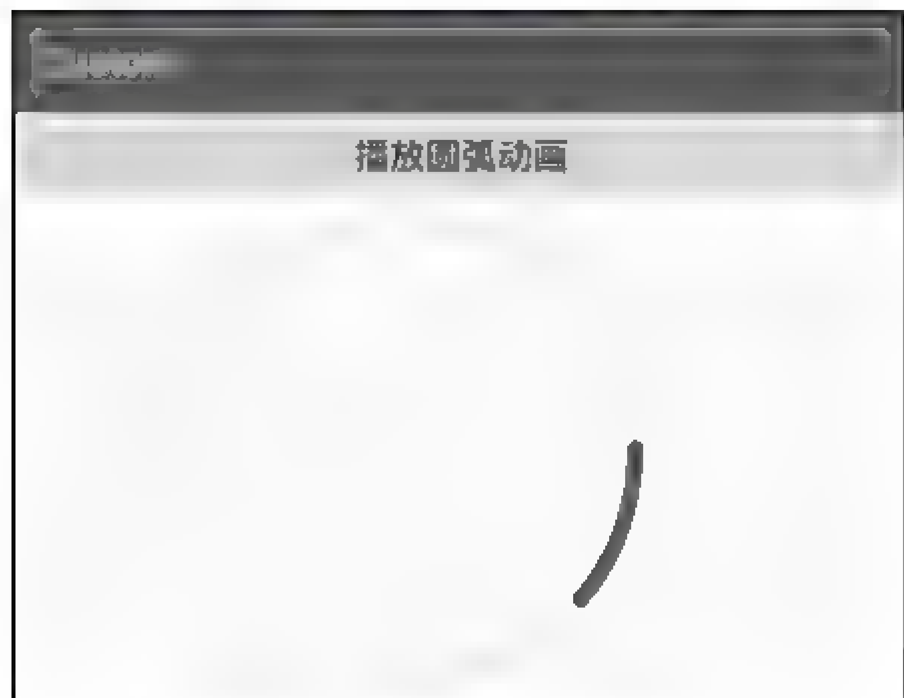


图 6-17　圆弧动画开始



图 6-18　圆弧动画结束

6.3　自定义对话框

本节介绍窗口与对话框的基本概念和使用方法，并利用基础对话框实现一个改进的日期选择对话框，结合第 5 章的列表视图与网格视图给出阶段性实战小项目——多级对话框的实现效果。

6.3.1　对话框 Dialog

App 界面附着在窗口 Window 上。大至整个活动页面，小至 Toast 的提示窗，还有对话框 Dialog，都建立在窗口上。如果想熟练掌握对话框，就必须先了解窗口。读者也许对窗口的概念不甚理解，下面从 Window 的 5 个常用方法开始介绍。

- setContentView：设置内容视图。这个方法是不是很熟悉？我们每天打交道的 Activity 第一句就是 setContentView，查看源码后发现内部原来调用了同名方法 getWindow().setContentView。
- setLayout：设置内容视图的宽、高尺寸。
- setGravity：设置内容视图的对齐方式。
- setBackgroundDrawable：设置内容视图的背景。
- findViewById：根据资源 ID 获取该视图的对象。这个方法每个 Activity 代码都要用许多遍。查看 Activity 源码后可以发现该方法也是调用 Window 的同名方法 getWindow().findViewById。

原来，窗口默默地做了许多事情，只是我们不知道罢了。熟悉了 Window 的概念和用法后，再来看看 Dialog 的工作机制，在屏幕上显示对话框主要有 3 个步骤：

步骤 01 构造一个对话框对象并指定该对话框的样式。

步骤 02 获取该对话框依赖的窗口对象，设置内容视图并指定窗口的尺寸。

步骤 03 完成相关属性设置，显示对话框。

下面来看具体的对话框操作方法。

- Dialog 构造函数：可定义对话框的主题样式（样式在 styles.xml 中定义），如是否有标题、是否为半透明、对话框的背景是什么等。
- getWindow：获取对话框的窗口对象。该方法是自定义对话框的关键，首先获取对话框所在的窗口对象，然后往这个窗口添加定制视图。
- show：显示对话框。
- isShowing：判断对话框是否显示。
- hide：隐藏对话框。
- dismiss：关闭对话框。
- setCancelable：设置对话框是否可取消。
- setCanceledOnTouchOutside：点击对话框外部区域是否自动关闭对话框。默认会自动关闭。
- setOnShowListener：设置对话框的显示监听器。需实现 OnShowListener 接口的 onShow 方法。
- setOnDismissListener：设置对话框的消失监听器。需实现 OnDismissListener 接口的 onDismiss 方法。

6.3.2 改进的日期对话框

对话框常用的一个控件是 AlertDialog，还有第 5 章介绍的 DatePickerDialog 和 TimePickerDialog。不过系统自带的对话框往往只能修改文字，无法调整界面布局，也无法定制按钮样式，甚至连文本的大小和颜色都无法修改。以日期选择对话框 DatePickerDialog 为例，该对话框的标题格式为“yyyy-mm-dd 周几”，按钮文字为“完成”，如图 6-19 所示。

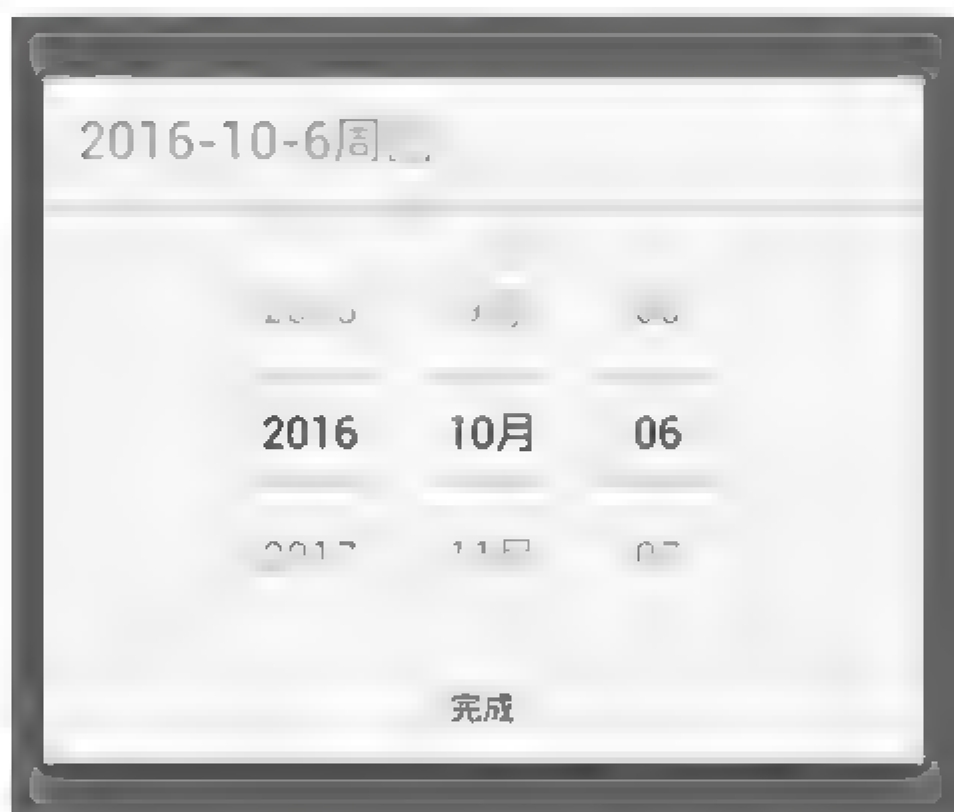


图 6-19 系统自带的日期对话框



图 6-20 改进后的日期对话框

这个对话框的标题文字无法修改，确定按钮的文字也无法修改。现在我们将这个日期对话框改头换面一下，使之更符合用户习惯，改造后的对话框效果如图 6-20 所示。其中，对话框标题改为“请选择日期”，确定按钮的文字也改为“确定”。具体的改造过程分 3 步：

01 定义一个对话框布局文件，在合适的地方放置标题文字的 TextView 控件、选择日期的 DatePicker 控件、确定按钮的 Button 控件等。详细的布局文件代码如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/transparent"
    android:gravity="center"
    android:orientation="vertical"
    android:paddingLeft="40dp"
    android:paddingRight="40dp" >

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical"
        android:background="@color/white" >

        <TextView
            android:id="@+id/tv_title"
            android:layout_width="match_parent"
            android:layout_height="60dp"
            android:paddingLeft="10dp"
            android:gravity="left|center"
            android:text="请选择日期"
            android:textColor="@color/blue"
            android:textSize="22sp" />

        <View
            android:layout_width="match_parent"
            android:layout_height="2dp"
            android:background="@color/blue" />

        <DatePicker
            android:id="@+id/dp_date"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:calendarViewShown="false"
            android:gravity="center"
            android:spinnersShown="true" />

        <View
            android:layout_width="match_parent"
```

```
                android:layout_height="1dp"
                android:background="@color/blue" />

            <Button
                android:id="@+id/btn_ok"
                android:layout_width="match_parent"
                android:layout_height="60dp"
                android:background="@null"
                android:gravity="center"
                android:text="确定"
                android:textColor="@color/black"
                android:textSize="17sp" />
        </LinearLayout>
</LinearLayout>
```

02 编写自定义日期对话框的代码，设置对话框的布局、样式、日期、标题，并处理确定按钮的点击事件、日期选择器的变更事件等。自定义对话框的完整代码如下：

```
public class CustomDateDialog implements OnClickListener, OnDateChangedListener {
    private Dialog dialog;
    private View view;
    private TextView tv_title;
    private DatePicker dp_date;

    public CustomDateDialog(Context context) {
        view = LayoutInflater.from(context).inflate(R.layout.dialog_date, null);
        dialog = new Dialog(context, R.style.CustomDateDialog);
        tv_title = (TextView) view.findViewById(R.id.tv_title);
        dp_date = (DatePicker) view.findViewById(R.id.dp_date);
        view.findViewById(R.id.btn_ok).setOnClickListener(this);
    }

    public void setTitle(String title) {
        tv_title.setText(title);
    }

    public void setDate(int year, int month, int day, OnDateSetListener listener) {
        dp_date.init(year, month, day, this);
        mDateSetListener = listener;
    }

    public void show() {
        dialog.getWindow().setContentView(view);
        dialog.getWindow().setLayout(LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
```



```
            dialog.show();
        }

        public void dismiss() {
            if (dialog != null && dialog.isShowing()) {
                dialog.dismiss();
            }
        }

        public boolean isShowing() {
            if (dialog != null) {
                return dialog.isShowing();
            } else {
                return false;
            }
        }

        @Override
        public void onClick(View v) {
            dismiss();
            if (mDateSetListener != null) {
                dp_date.clearFocus();
                //这里给系统月份加一，调用时就不必每次都加一了
                mDateSetListener.onDateSet(dp_date.getYear(),
                        dp_date.getMonth()+1, dp_date.getDayOfMonth());
            }
        }

        @Override
        public void onDateChanged(DatePicker view, int year, int monthOfYear, int dayOfMonth) {
            dp_date.init(year, monthOfYear, dayOfMonth, this);
        }

        private OnDateSetListener mDateSetListener;
        public interface OnDateSetListener {
            void onDateSet(int year, int monthOfYear, int dayOfMonth);
        }
}
```

03 在 Acitvity 页面中使用自定义的日期对话框，调用代码举例如下：

```
        Calendar calendar = Calendar.getInstance();
        CustomDateDialog dialog = new CustomDateDialog(this);
        dialog.setDate(calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH),
```

```
                calendar.get(Calendar.DAY_OF_MONTH), this);
        dialog.show();
```

6.3.3 自定义多级对话框

在实际开发中，自定义对话框往往比较复杂，比如对话框不在屏幕中央而在屏幕下方、对话框在消失前还需做其他处理、对话框像多级菜单一样需要分级展示等。

如图 6-21 与图 6-22 所示，选择对话框分两级显示。图 6-21 展示的是好友列表，用到了列表视图 ListView；图 6-22 展示的是好友关系，用到了网格视图 GridView，二级对话框位于一级对话框之上。



图 6-21 第一级对话框



图 6-22 第二级对话框

这个多级对话框是一个阶段性的实战小项目，不但运用了自定义对话框的进阶实现，而且使用了第 5 章介绍的列表视图与网格视图，有兴趣的读者可尝试将其编码实现。完整的多级对话框代码参见本书的下载资源。

6.4 自定义通知栏

本节介绍通知栏的用法和如何自定义通知栏，包括通知推送 Notification 的设置、进度条 ProcessBar 的样式定制、远程视图 RemoteViews 的配置方法，并给出一个自定义通知栏的具体例子。

6.4.1 通知推送 Notification

在手机屏幕的顶端下拉会弹出通知栏，里面存放的是 App 即时提醒用户的消息，消息内容由 Notification 产生并推送。每条消息通知基本都有图标、标题、内容、时间等元素，参数通过 Notification.Builder 构建。下面来看常用的参数构建方法。

- setWhen：设置推送时间，格式为“小时：分钟”。推送时间在通知栏右方显示。
- setShowWhen：设置是否显示推送时间。

- setUsesChronometer：设置是否显示计数器。为 true 时不显示推送时间，动态显示从通知被推送到当前的时间间隔，以“分钟：秒钟”格式显示。
- setSmallIcon：设置状态栏里面的图标（小图标）。
- setTicker：设置状态栏里面的提示文本。
- setLargeIcon：设置通知栏里面的图标（大图标）。
- setContentTitle：设置通知栏里面的标题文本。
- setContentText：设置通知栏里面的内容文本。
- setSubText：设置通知栏里面的附加说明文本，位于内容文本下方。若调用该方法，则 setProgress 的设置失效。
- setProgress：设置进度条与当前进度。进度条位于标题文本与内容文本中间。
- setNumber：设置通知栏右下方的数字，可与 setProgress 联合使用，表示当前的进度数值。
- setContentInfo：设置通知栏右下方的文本。若调用该方法，则 setNumber 的设置失效。
- setContentIntent：设置内容的延迟意图 PendingIntent，点击该通知时触发该意图。通常调用 PendingIntent 的 getActivity 方法获得延迟意图对象，getActivity 表示点击后跳转到该页面。
- setDeleteIntent：设置删除的延迟意图 PendingIntent，滑掉该通知时触发该动作。
- setAutoCancel：设置该通知是否自动清除。若为 true，则点击该通知后，通知会自动消失；若为 false，则点击该通知后，通知不会消失。
- setContent：设置一个定制的通知栏视图 RemoteViews，用于取代 Builder 的默认视图模板。
- build：构建方法。在以上参数都设置完毕后，调用该方法返回 Notification 对象。

使用以上设置方法要注意 4 点：

（1）setSmallIcon 方法必须调用，否则不会显示通知消息。

（2）setWhen 与 setUsesChronometer 同时只能调用其中一个，即推送时间与计数器无法同时显示，因为它们都位于通知栏右边。

（3）setSubText 与 setProgress 同时只能调用其中一个，因为附加说明与进度条都位于标题文本下方。

（4）setNumber 与 setContentInfo 同时只能调用其中一个，因为计数值与提示都位于通知栏右下方。

使用 Notification 只能生成通知内容，实际推送动作还需借助系统的通知服务实现。NotificationManager 是系统通知服务的管理类，有以下 3 个常用方法。

- notify：推送指定消息到通知栏。
- cancel：取消指定消息。调用该方法后，通知栏中的指定消息将消失。
- cancelAll：取消所有消息。

下面是发送简单消息的代码片段：

```
    private void sendSimpleNotify(String title, String message) {
        Intent clickIntent = new Intent(this, MainActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this,
```

```
                    R.string.app_name, clickIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        Notification.Builder builder = new Notification.Builder(this);
        builder.setContentIntent(contentIntent)
                    .setAutoCancel(true).setSmallIcon(R.drawable.ic_app)
                    .setTicker("提示消息来啦").setWhen(System.currentTimeMillis())
                    .setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.ic_app))
                    .setContentTitle(title).setContentText(message);
        Notification notify = builder.build();
        NotificationManager notifyMgr = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        notifyMgr.notify(R.string.app_name, notify);
    }
```

简单消息的通知栏效果如图 6-23 所示，左边是图标，中间是标题与内容，右边是时间。



图 6-23 简单消息的通知栏效果

下面是发送计时消息的代码片段：

```
    private void sendCounterNotify(String title, String message) {
        Intent cancelIntent = new Intent(this, MainActivity.class);
        PendingIntent deleteIntent = PendingIntent.getActivity(this,
                    R.string.app_name, cancelIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        Notification.Builder builder = new Notification.Builder(this);
        builder.setDeleteIntent(deleteIntent)
                    .setAutoCancel(true).setUsesChronometer(true)
                    .setProgress(100, 60, false).setNumber(99)
                    .setSmallIcon(R.drawable.ic_app).setTicker("提示消息来啦")
                    .setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.ic_app))
                    .setContentTitle(title).setContentText(message);
        Notification notify = builder.build();
        NotificationManager notifyMgr = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        notifyMgr.notify(R.string.app_name, notify);
    }
```

计时消息的通知栏效果如图 6-24 所示，通知栏左边是图标，中间是标题文本、进度条和内容文本，右边是计时器与计数值。

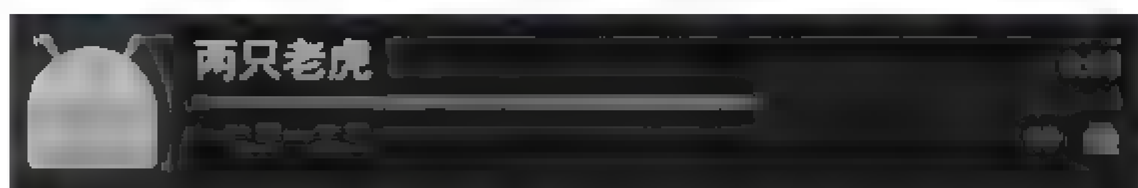


图 6-24 计时消息的通知栏效果

6.4.2 进度条 ProcessBar

消息通知 Notification 的 setProgress 方法是对内置进度条进行操作，不过很多时候进度条会单独使用，有必要了解一下 ProcessBar 的具体用法。

下面来看进度条的常用属性。

- style：指定进度条的形状样式。?android:attr/progressBarStyleHorizontal 表示水平形状，?android:attr/progressBarStyle 表示圆圈形状。
- max：指定进度条的最大值。
- progress：指定进度条当前进度值。
- secondaryProgress：指定进度条当前次要进度值。比如播放视频，progress 用来表示当前播放进度，secondaryProgress 用来表示当前缓冲进度。
- progressDrawable：指定进度条的进度图形。

进度条的常用方法有以下 9 个。

- setProgress：设置当前进度。
- getProgress：获取当前进度。
- setSecondaryProgress：设置次要进度。
- getSecondaryProgress：获取次要进度。
- setMax：设置进度条的最大值。
- getMax：获取进度条的最大值。
- incrementProgressBy：设置当前进度的增量。
- incrementSecondaryProgressBy：设置次要进度的增量。
- setProgressDrawable：设置进度条的进度图形。

使用进度条时需要注意以下两点：

（1）max、progress 的相关属性和方法只在样式为 progressBarStyleHorizontal 时才有效，即水平进度条可动态设置进度值；如果样式为 progressBarStyle 圆圈形状，最大值与进度值的设置就会失效，即圆圈形状不会显示当前进度，只会兀自旋转。想实现动态显示进度的进度条，可参考 6.2 节的圆弧进度动画。

（2）progressDrawable 进度图形不能用普通图形，只能用层次图形 LayerDrawable。层次图形可在 XML 文件中定义，如果用于描述进度图形就要同时定义两个层次，即背景层次与进度条层次。例如，在自定义圆弧动画时运用了背景视图与前景视图，在进度条中就存在背景层次，只不过前景视图换成了进度条层次。

下面是一个层次图形定义的 XML 例子。其中，根节点 layer-list 表示这是一个层次列表，即层次图形定义；背景层次的 id 为@android:id/background，采用的是形状图形（节点名称为 shape）；进度条层次的 id 为@android:id/progress，采用的是裁剪图形 ClipDrawable（节点名称为 clip）：

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@android:id/background">
        <shape>
            <solid android:color="#333333" />
        </shape>
    </item>
    <item android:id="@android:id/progress">
        <clip>
            <nine-patch android:src="@drawable/notify_green" />
        </clip>
    </item>
</layer-list>
```

下面是进度条控件在布局文件中使用的 XML 代码：

```
    <ProgressBar
        android:id="@+id/pb_progress"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="30dp"
        android:background="@color/black"
        android:max="100"
        android:progress="0"
        android:progressDrawable="@drawable/notify_progress_green" />
```

进度条设置前后的效果如图 6-25 与图 6-26 所示。其中，图 6-25 所示为进度值为 0 的界面，此时只有一条黑色的进度条背景；图 6-26 所示为进度值为 40 的界面，此时绿色进度条占据全部进度长度的 40%。



图 6-25　进度值为 0 的进度条

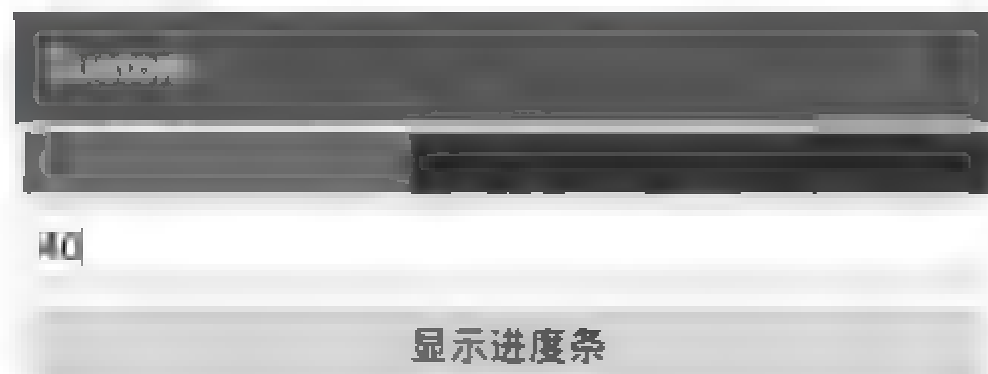


图 6-26　进度值为 40 的进度条

6.4.3　远程视图 RemoteViews

前面介绍 Notification 的常用方法时提到 setContent 方法可以在设置定制的通知栏视图 RemoteViews 时取代 Builder 的默认视图模板。这表示通知栏允许自定义，并且自定义通知栏需要采用远程视图 RemoteViews。

与活动页面相比，如果说对话框是一个小型页面，远程视图就是一个小型且简化的页面。简化的意思是功能减少了，限制变多了。虽然 RemoteViews 与 Activity 一样有自己的布局文件，但是 RemoteViews 的使用权限小了很多。两者的区别主要有：

（1）RemoteViews 主要用于通知栏部件和桌面部件，而 Activity 用于页面。

（2）RemoteViews 只支持少数几种控件，如 TextView、ImageView、Button、ImageButton、ProcessBar、Chronometer（计时器）和 AnalogClock（模拟时钟）。

（3）RemoteViews 不可直接获取和设置控件信息，只能通过该对象的 set 方法修改控件信息。

下面来看远程视图的常用方法。

- 构造函数：创建一个 RemoteViews 对象。第一个参数是包名，第二个参数是布局文件 id。
- setViewVisibility：设置指定控件是否可见。
- setViewPadding：设置指定控件的间距。
- setTextViewText：设置指定 TextView 或 Button 控件的文字内容。
- setTextViewTextSize：设置指定 TextView 或 Button 控件的文字大小。
- setTextColor：设置指定 TextView 或 Button 控件的文字颜色。
- setTextViewCompoundDrawables：设置指定 TextView 或 Button 控件的文字周围图标。
- setImageViewResource：设置 ImageView 或 ImgaeButton 控件的资源编号。
- setImageViewBitmap：设置 ImageView 或 ImgaeButton 控件的位图对象。
- setChronometer：设置计时器信息。
- setProgressBar：设置进度条信息，包括最大值与当前进度。
- setOnClickPendingIntent：设置指定控件的点击响应动作。

完成 RemoteViews 对象的构建与设置后调用 Notification 对象的 setContent 方法，即可完成自定义通知的定义。

下面是一个远程视图用到的布局文件代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:minHeight="64dp"
    android:orientation="horizontal" >

    <ImageView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:scaleType="fitCenter"
        android:src="@drawable/tt" />

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp"
        android:layout_weight="5"
```

```
        android:orientation="vertical" >

        <ProgressBar
            android:id="@+id/pb_play"
            style="?android:attr/progressBarStyleHorizontal"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:max="100"
            android:progress="10" />

        <TextView
            android:id="@+id/tv_play"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:textColor="#ffffff"
            android:textSize="17sp" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:orientation="vertical" >

        <Chronometer
            android:id="@+id/chr_play"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1" />

        <Button
            android:id="@+id/btn_play"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="2"
            android:text="暂停"
            android:textColor="#ffffff"
            android:textSize="17sp" />
    </LinearLayout>
</LinearLayout>
```

下面是自定义通知栏的调用代码：

```
    private void sendCustomNotify(Context ctx, String event, String song, boolean isPlay, int progress, long
time) {
```

```
            Intent pIntent = new Intent(event);
            PendingIntent nIntent = PendingIntent.getBroadcast(
                    ctx, R.string.app_name, pIntent, PendingIntent.FLAG_UPDATE_CURRENT);
            RemoteViews notify_music = new RemoteViews(ctx.getPackageName(), R.layout.notify_music);
            if (isPlay == true) {
                notify_music.setTextViewText(R.id.btn_play, "暂停");
                notify_music.setTextViewText(R.id.tv_play, song+"正在播放");
                notify_music.setChronometer(R.id.chr_play, time, "%s", true);
            } else {
                notify_music.setTextViewText(R.id.btn_play, "继续");
                notify_music.setTextViewText(R.id.tv_play, song+"暂停播放");
                notify_music.setChronometer(R.id.chr_play, time, "%s", false);
            }
            notify_music.setProgressBar(R.id.pb_play, 100, progress, false);
            notify_music.setOnClickPendingIntent(R.id.btn_play, nIntent);
            Intent intent = new Intent(ctx, MainActivity.class);
            PendingIntent contentIntent = PendingIntent.getActivity(ctx,
                    R.string.app_name, intent, PendingIntent.FLAG_UPDATE_CURRENT);
            Notification.Builder builder = new Notification.Builder(ctx);
            builder.setContentIntent(contentIntent)
                    .setContent(notify_music).setTicker(song).setSmallIcon(R.drawable.tt_s);
            Notification notify = builder.build();
            NotificationManager notifyMgr = (NotificationManager) getSystemService
(Context.NOTIFICATION_SERVICE);
            notifyMgr.notify(R.string.app_name, notify);
        }
```

自定义通知栏的效果如图 6-27 所示，可以看到播放器图标在通知栏左边，进度条在上方，歌曲名称在下方，计时器与控制按钮分布在通知栏右边。



图 6-27　自定义通知栏的效果图

6.5　Service 基础

本节介绍为何使用服务 Service 和如何使用服务，包括服务的生命周期和在 3 种启停方式下的生命周期过程，有普通启停、立即绑定和延迟绑定。另外，介绍怎样结合通知推送 Notification 实现把服务推送到前台的功能。



6.5.1 Service 的生命周期

服务 Service 是 Android 的四大组件之一，常用在看不见页面的高级场合，如第 5 章定时器用到了系统的闹钟服务，6.4 节通知推送用到了系统的通知服务。既然 Android 有系统服务，App 也可以有自己的服务。Service 与 Activity 相比，不同之处在于没有对应的页面，相同之处在于有生命周期。要想用好服务，就要探究其生命周期。

下面是 Service 与生命周期有关的方法说明。

- onCreate：创建服务。
- onStart：开始服务，Android2.0 以下版本使用，现已废弃。
- onStartCommand：开始服务，Android2.0 及以上版本使用。该函数的返回值说明见表 6-5。

表 6-5 服务启动的返回值说明

返回值类型	返回值说明
START_STICKY	粘性的服务。如果服务进程被杀掉，就保留服务的状态为开始状态，但不保留传送的 Intent 对象。随后系统尝试重新创建服务，由于服务状态为开始状态，因此创建服务后一定会调用 onStartCommand 方法。如果在此期间没有任何启动命令传送给服务，参数 Intent 就为空值
START_NOT_STICKY	非粘性的服务。使用这个返回值时，如果服务被异常杀掉，系统就不会自动重启该服务
START_REDELIVER_INTENT	重传 Intent 的服务。使用这个返回值时，如果服务被异常杀掉，系统就会自动重启该服务，并传入 Intent 的原值
START_STICKY_COMPATIBILITY	START_STICKY 的兼容版本，不保证服务被杀掉后一定能重启

- onDestroy：销毁服务。
- onBind：绑定服务。
- onRebind：重新绑定。该方法只有当上次 onUnbind 返回 true 的时候才会被调用。
- onUnbind：解除绑定。返回值为 true 表示允许再次绑定，再绑定时调用 onRebind 方法；返回值为 false 表示只能绑定一次，不能再次绑定，默认为 false。

看来 Service 的生命周期也不简单，分好几种生命周期方法。原因是服务存在多种启停方式，如普通启停、立即绑定、延迟绑定，每种启停方式都对应不同的周期方法。下面分别叙述 3 种启停方式及其生命周期说明。

1. 普通启停

普通启停是最简单的用法。下面是该方式的服务代码：

```
public class SimpleService extends Service {
    private static final String TAG = "SimpleService";

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
```



```
            Log.d(TAG, "测试服务到此一游！");
            return START_STICKY;
        }

        @Override
        public IBinder onBind(Intent intent) {
            return null;
        }
}
```

这个服务很简单，功能只是打印一行日志“测试服务到此一游！”。在 Acitivity 代码中，启停服务也很简单，调用 startService 方法即可启动服务，调用 stopService 方法即可停止服务。当然，也可以在 Intent 对象中传递参数信息。示例的调用代码如下：

```
            Intent intent = new Intent(this, SimpleService.class);
            startService(intent);
```

普通启停方式的服务生命周期可通过打印日志观察，也可在页面上直接显示日志。启动服务依次调用了 onCreate 与 onStartCommand 方法，如图 6-28 所示。停止服务调用了 onDestroy 方法，如图 6-29 所示。

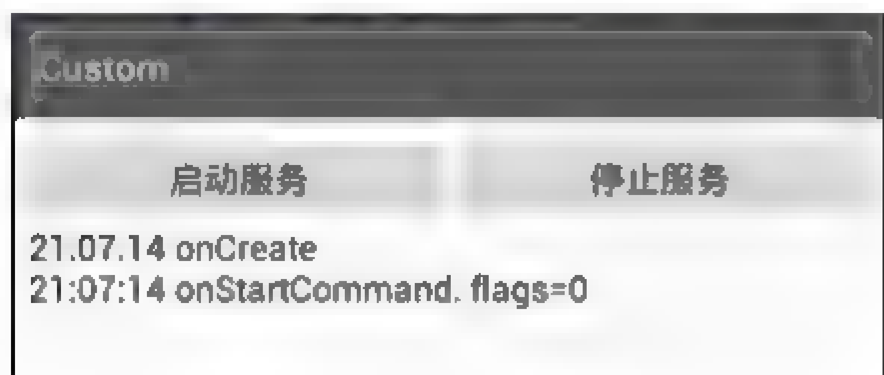


图 6-28　启动服务的日志

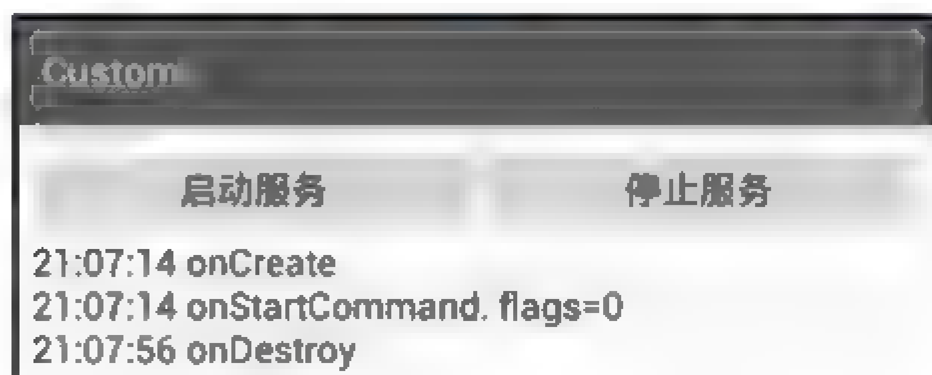


图 6-29　停止服务的日志

2. 立即绑定

绑定方式的服务定义有所不同，因为绑定的服务可能运行于另一个进程，所以必须定义一个 Binder 对象用来进行进程间的通信。下面是一个绑定方式的服务代码：

```
public class BindImmediateService extends Service {
        private static final String TAG = "BindImmediateService";
        private final IBinder mBinder = new LocalBinder();
        public class LocalBinder extends Binder {
            public BindImmediateService getService() {
                return BindImmediateService.this;
            }
        }

        @Override
        public IBinder onBind(Intent intent) {
            Log.d(TAG, "绑定服务开始旅程！");
```

```
            return mBinder;
        }

        @Override
        public boolean onUnbind(Intent intent) {
            Log.d(TAG, "绑定服务结束旅程！");
            return false;
        }
    }
```

这个服务在绑定时会打印日志"绑定服务开始旅程！"，在解除绑定时会打印日志"绑定服务结束旅程！"。在 Activity 中，绑定/解绑服务的做法与普通方式不同，首先要定义一个 ServiceConnection 的服务连接对象，然后调用 bindService 方法或 unbindService 方法进行绑定或解绑操作，具体的示例代码如下：

```
    public class BindImmediateActivity extends AppCompatActivity implements OnClickListener {
        private static TextView tv_immediate;
        private Intent mIntent;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_bind_immediate);
            tv_immediate = (TextView) findViewById(R.id.tv_immediate);
            findViewById(R.id.btn_start_bind).setOnClickListener(this);
            findViewById(R.id.btn_unbind).setOnClickListener(this);
            mIntent = new Intent(this, BindImmediateService.class);
        }

        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.btn_start_bind) {
                boolean bindFlag = bindService(mIntent, mFirstConn, Context.BIND_AUTO_CREATE);
            } else if (v.getId() == R.id.btn_unbind) {
                if (mBindService != null) {
                    unbindService(mFirstConn);
                    mBindService = null;
                }
            }
        }

        private BindImmediateService mBindService;
        private ServiceConnection mFirstConn = new ServiceConnection() {
            //获取服务对象时的操作
```

```
            public void onServiceConnected(ComponentName name, IBinder service) {
                //如果服务运行于另一个进程，就不能直接强制转换类型
                //否则会报错 java.lang.ClassCastException：android.os.BinderProxy cannot be cast to...
                mBindService = ((BindImmediateService.LocalBinder) service).getService();
            }

            //无法获取服务对象时的操作
            public void onServiceDisconnected(ComponentName name) {
                mBindService = null;
            }
        };
}
```

接下来，继续观察立即绑定方式的生命周期，该方式的服务周期日志如图 6-30 和图 6-31 所示。其中，图 6-30 所示为立即绑定时的界面，此时依次调用 onCreate 和 onBind 方法；图 6-31 所示为立即解绑时的界面，此时依次调用 onUnbind 和 onDestroy 方法。



图 6-30　立即绑定的日志



图 6-31　立即解绑的日志

3. 延迟绑定

延迟绑定与立即绑定的区别在于：延迟绑定是在页面上先通过 startService 方法启动服务，然后通过 bindService 方法绑定已存在的服务。这样一来，因为启动操作在先，所以解绑操作只能撤销绑定操作，而不能撤销启动操作。由于解绑服务不能停止服务，因此存在再次绑定服务的可能。

下面观察延迟绑定的日志，验证一下实际结果是否符合我们的猜想。依次查看“启动服务→绑定服务→解绑服务”的运行日志，如图 6-32 所示；依次查看“绑定服务→解绑服务→停止服务”的运行日志，如图 6-33 所示。



图 6-32　延迟绑定的日志



图 6-33　再次绑定的日志

从日志中可以看到，延迟绑定与立即绑定两种方式的生命周期区别在于：

（1）延迟绑定的首次绑定操作只调用 onBind 方法，再次绑定只调用 onRebind 方法（是否允许再次绑定要看上次 onUnbind 方法的返回值）。

（2）延迟绑定的解绑操作只调用 onUnbind 方法。

6.5.2 推送服务到前台

服务没有自己的布局文件，也就意味着无法直接在页面上展示，要想了解服务的运行情况，要么通过打印日志，要么获取某个页面的静态对象，然后在该页面上显示运行结果。然而活动页面有自身的生命周期，极有可能发生服务尚在运行但页面早已退出的情况，所以该方式不可靠。幸好，服务不只能在外部进行启停或绑定，还能在内部模拟启停，当然仅是模拟而已。

服务内部的启停方法也有对应的两个函数。

- startForeground：把当前服务切换到前台运行。第一个参数表示通知的编号，第二个参数表示 Notification 对象，意味着切换到前台就是展示到通知栏。
- stopForeground：停止前台运行。参数为 true 表示清除通知，参数为 false 表示不清除。

服务在前台运行的一个常见的应用是音乐播放器，即使用户离开了播放器页面，手机仍然能在后台继续播放音乐，同时还能在通知栏查看播放进度，控制播放与暂停操作。下面是一个音乐播放服务的例子：

```
@TargetApi(Build.VERSION_CODES.JELLY_BEAN)
public class MusicService extends Service {
    private final IBinder mBinder = new LocalBinder();
    public class LocalBinder extends Binder {
        public MusicService getService() {
            return MusicService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        Log.d(TAG, "onBind");
        return mBinder;
    }

    private String mSong;
    private String PAUSE_EVENT = "";
    private boolean bPlay = true;
    private long mBaseTime;
    private long mPauseTime = 0;
    private int mProcess = 0;
    private Handler mHandler = new Handler();
```

```
    private Runnable mPlay = new Runnable() {
        @Override
        public void run() {
            if (bPlay == true) {
                mProcess = (mProcess < 100) ? mProcess+2 : 0;
                mHandler.postDelayed(this, 1000);
            }
            Notification notify = getNotify(MusicService.this, PAUSE_EVENT, mSong, bPlay,
mProcess, mBaseTime);
            startForeground(2, notify);
        }
    };

    private Notification getNotify(Context ctx, String event, String song, boolean isPlay, int progress, long
time) {
        Intent pIntent = new Intent(event);
        PendingIntent nIntent = PendingIntent.getBroadcast(
                ctx, R.string.app_name, pIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        RemoteViews notify_music = new RemoteViews(ctx.getPackageName(), R.layout.notify_music);
        if (isPlay == true) {
            notify_music.setTextViewText(R.id.btn_play, "暂停");
            notify_music.setTextViewText(R.id.tv_play, song+"正在播放");
            notify_music.setChronometer(R.id.chr_play, time, "%s", true);
        } else {
            notify_music.setTextViewText(R.id.btn_play, "继续");
            notify_music.setTextViewText(R.id.tv_play, song+"暂停播放");
            notify_music.setChronometer(R.id.chr_play, time, "%s", false);
        }
        notify_music.setProgressBar(R.id.pb_play, 100, progress, false);
        notify_music.setOnClickPendingIntent(R.id.btn_play, nIntent);
        Intent intent = new Intent(ctx, MainActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(ctx,
                R.string.app_name, intent, PendingIntent.FLAG_UPDATE_CURRENT);
        Notification.Builder builder = new Notification.Builder(ctx);
        builder.setContentIntent(contentIntent).setContent(notify_music)
                .setTicker(song).setSmallIcon(R.drawable.tt_s);
        Notification notify = builder.build();
        return notify;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startid) {
        mBaseTime = SystemClock.elapsedRealtime();
```

```
            bPlay = intent.getBooleanExtra("is_play", true);
            mSong = intent.getStringExtra("song");
            mHandler.postDelayed(mPlay, 200);
            return START_STICKY;
        }

        @Override
        public void onCreate() {
            PAUSE_EVENT = getResources().getString(R.string.pause_event);
            pauseReceiver = new PauseReceiver();
            IntentFilter filter = new IntentFilter(PAUSE_EVENT);
            registerReceiver(pauseReceiver, filter);
            super.onCreate();
        }

        @Override
        public void onDestroy() {
            unregisterReceiver(pauseReceiver);
            super.onDestroy();
        }

        private PauseReceiver pauseReceiver;
        public class PauseReceiver extends BroadcastReceiver {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent != null) {
                    bPlay = !bPlay;
                    if (bPlay == true) {
                        mHandler.postDelayed(mPlay, 200);
                        if (mPauseTime > 0) {
                            long gap = SystemClock.elapsedRealtime() - mPauseTime;
                            mBaseTime += gap;
                        }
                    } else {
                        mPauseTime = SystemClock.elapsedRealtime();
                    }
                }
            }
        }
    }
```

上述代码的与众不同之处在于点击播放/暂停按钮的处理，此时触发的延迟意图对象由getBroadcast方法获得，原因是getActivity获得的对象只会跳转到某个页面，要想让触发的事件作用于服务内部，只能通过广播的方式。

音乐播放服务的前台运行效果如图 6-34 和图 6-35 所示。其中，图 6-34 所示为正在播放的通知栏界面，图 6-35 所示为暂停播放的通知栏界面。



图 6-34　正在播放的通知栏界面



图 6-35　暂停播放的通知栏界面

6.6　实战项目：手机安全助手

本节将设计一个实战项目——手机安全助手，该项目采用多种自定义控件的相关技术，并同时运用多种存储技术。通过该实战项目的练习能够加深自定义控件的用法理解，还能复习巩固前两章的存储技术知识。

6.6.1　设计思路

如同电脑上的杀毒软件，手机上也有形形色色的安全 App，比如**安全管家、**安全卫士、**安全助手等，这些安全 App 都有一个核心模块——流量监控功能。现在运营商都靠流量赚钱，比如 100M 流量要 10 元钱、1G 流量要 100 元，很多 App 一打开就是满屏图片，非常费流量，而且有的 App 会偷跑流量，很多用户不知不觉电话费就被流量花光了。流量监控功能很实用，基础实现也不难，下面就以流量监控为例，做一个“手机安全助手”的实战项目，活学活用本章自定义控件的相关知识。

先来看手机安全助手的总体页面效果。为了起到提醒作用，对于超出限额的流量部分使用含有警示意义的橙色显示，如图 6-36 所示。如果当天已用流量未超出限额，就使用绿色显示流量信息，如图 6-37 所示。



图 6-36　流量限额为 30M 的流量页面



图 6-37　流量限额为 50M 的流量页面

上面说的流量限额可在配置页面进行设置，配置页面效果如图 6-38 所示。



图 6-38　流量限额配置页面

再来看助手 App 的通知栏效果，超出限额的流量同样使用橙色进度条展示，如图 6-39 所示；若未超出当日限额，则使用绿色展示进度条，如图 6-40 所示。



图 6-39　流量限额为 30M 的通知栏



图 6-40　流量限额为 50M 的通知栏

下面来看这个安全助手用到了本章哪些新技术，机灵的你一定不会错过以下 4 点。

- 自定义日期对话框：最上面的标题栏，统计日期的选择对话框可采用自定义形式。
- 自定义圆弧动画：页面上方的流量信息，使用圆弧动画展示当天的已用流量，并通过圆弧颜色提醒当前流量是否超标。
- 自定义通知栏：通知栏中包含定制样式的进度条，必须采用自定义通知栏。
- 服务 Service：流量数据每间隔一段时间就得重新获取，这种定时处理无法在 Acitivity 页面进行，只能在服务 Service 中处理。

另外，安全助手还运用了多种存储技术，下面一一道来。

- 数据库：毫无疑问，历史流量数据必须保存在数据库中。
- 共享参数：每日的流量限额可直接保存在共享参数中。
- 全局内存：也许读者不理解这里跟全局内存有什么关系，其实全局内存要保存数据库连接，因为主页面需要通过数据库查询流量数据，后台服务也要不断获取流量数据并更新至数据库，既然不止一个地方用到数据库连接，不如统一放到全局内存中，还可以避免因资源释放造成别处访问不了的异常。

如此看来，该实战项目不但可以演练各种自定义控件，而且可以复习前两章的数据存储技术，可谓一举两得。

6.6.2　小知识：应用包管理 PackageManager

手机安全管理涉及获取已安装应用的应用包信息，包括应用的进程编号、名称、图标以及流量信息。其中，应用包的基本信息可通过 PackageManager 与 ApplicationInfo 联合获得，应用包信息的获取代码如下：

```
	public static ArrayList<AppInfo> getAppInfo(Context ctx, int type) {
		ArrayList<AppInfo> appList = new ArrayList<AppInfo>();
		SparseIntArray siArray = new SparseIntArray();
		PackageManager pm = ctx.getPackageManager();
		List<ApplicationInfo> installList = pm.getInstalledApplications(PackageManager.
PERMISSION_GRANTED);
		for (int i=0; i<installList.size(); i++) {
			ApplicationInfo item = installList.get(i);
			if (siArray.indexOfKey(item.uid) >= 0) {   //去掉重复的应用信息
				continue;
			}
			siArray.put(item.uid, 1);
			try {
				String[] permissions = pm.getPackageInfo(item.packageName, PackageManager.
GET_PERMISSIONS).requestedPermissions;
				if (permissions == null) {
					continue;
				}
				boolean bNet = false;
				for (String permission : permissions) {
					if (permission.equals("android.permission.INTERNET")) {
						bNet = true;
						break;
					}
				}
				if (type==0 || (type==1 && bNet)) {
					AppInfo app = new AppInfo();
					app.uid = item.uid;
					app.label = item.loadLabel(pm).toString();
					app.package_name = item.packageName;
					app.icon = item.loadIcon(pm);
					appList.add(app);
				}
			} catch (Exception e) {
				e.printStackTrace();
				continue;
			}
		}
		return appList;
	}
```

应用产生的流量数据可通过工具类 TrafficStats 读取，该工具有以下 6 种常用方法。



- getTotalRxBytes：获取接收流量的总字节数。
- getTotalTxBytes：获取发送流量的总字节数。
- getMobileRxBytes：获取数据连接接收流量的总字节数。包含移动数据流量，不含 wifi 流量。
- getMobileTxBytes：获取数据连接发送流量的总字节数。
- getUidRxBytes：获取指定进程接收流量的总字节数。
- getUidTxBytes：获取指定进程发送流量的总字节数。

6.6.3 代码示例

本章的实战项目依然要考虑代码架构，故而编码过程与第 5 章一样分为 5 步。

步骤 01 设计代码架构，初步拆分后的 package 包分为以下 7 部分。

- com.example.assistant.activity：存放 Acitivity 页面的代码。
- com.example.assistant.adapter：存放适配器的代码。
- com.example.assistant.bean：存放实体数据结构的代码，如日流量的字段信息。
- com.example.assistant.database：存放读写 SQLite 的数据库操作代码。
- com.example.assistant.service：存放服务 Service 的代码。
- com.example.assistant.util：存放工具类的代码。
- com.example.assistant.widget：存放自定义控件的代码。

步骤 02 想好代码文件与布局文件的名称，比如流量主页面的代码文件取名 MobileAssistantActivity.java，对应的布局文件名是 activity_mobile_assistant.xml；限额设置页面的代码文件取名 MobileConfigActivity.java，对应的布局文件名是 activity_mobile_config.xml。不要忘了流量统计服务的代码文件 TrafficService.java，还有适配器、对话框、远程视图的代码及其布局文件，读者可自行构思。

步骤 03 在 AndroidManifest.xml 中补充相应配置，主要有以下 3 点。

（1）注册两个页面的 acitivity 节点，注册代码如下：

```
<activity android:name=".MobileAssistantActivity" />
<activity android:name=".MobileConfigActivity" />
```

（2）注册流量统计服务的 service 节点，注册代码如下：

```
<service android:name=".service.TrafficService" android:enabled="true" />
```

（3）给 application 补充 name 属性，值为 MainApplication，举例如下：

```
android:name=".MainApplication"
```

步骤 04 在资源目录下补充相应的 XML 配置。

（1）在 res/drawable 目录加入定制进度条需要的层次图形描述文件。
（2）在 res/layout 目录下编写页面、适配器、对话框、远程视图对应的布局文件。
（3）在 res/values/styles.xml 中补充自定义日期对话框的样式定义。

05 进行 java 代码开发，包括对页面、适配器、对话框、后台服务等进行编码。

下面是流量统计服务 TrafficService.java 的完整代码：

```
@TargetApi(Build.VERSION_CODES.JELLY_BEAN)
public class TrafficService extends Service {
	private static final String TAG = "TrafficService";
	private MainApplication app;
	private int limit_day;
	private int mNowDay;

	@Override
	public int onStartCommand(Intent intent, int flags, int startid) {
		app = MainApplication.getInstance();
		limit_day = Integer.parseInt(SharedUtil.getIntance(this).readShared("limit_day", "30"));
		mHandler.post(mRefresh);
		return START_STICKY;
	}

	private Handler mHandler = new Handler();
	private Runnable mRefresh = new Runnable() {
		@Override
		public void run() {
			refreshData();
			refreshNotify();
			mHandler.postDelayed(this, 10000);
		}
	};

	private void refreshData() {
		mNowDay = Integer.parseInt(DateUtil.getNowDateTime("yyyyMMdd"));
		//获取最新的流量信息
		ArrayList<AppInfo> appinfoList = AppUtil.getAppInfo(this, 1);
		for (int i=0; i<appinfoList.size(); i++) {
			AppInfo item = appinfoList.get(i);
			item.traffic = TrafficStats.getUidRxBytes(item.uid);
			item.month = mNowDay/100;
			item.day = mNowDay;
			appinfoList.set(i, item);
		}
		app.mTrafficHelper.insert(appinfoList);
	}

	private void refreshNotify() {
		String last_date = DateUtil.getAddDate(""+mNowDay, -1);
		ArrayList<AppInfo> lastArray = app.mTrafficHelper.query("day="+last_date);
```

```
        ArrayList<AppInfo> thisArray = app.mTrafficHelper.query("day="+mNowDay);
        long traffic_day = 0;
        for (int i=0; i<thisArray.size(); i++) {
            AppInfo item = thisArray.get(i);
            for (int j=0; j<lastArray.size(); j++) {
                if (item.uid == lastArray.get(j).uid) {
                    item.traffic -= lastArray.get(j).traffic;
                    break;
                }
            }
            traffic_day += item.traffic;
        }
        String desc = "今日已用流量" + StringUtil.formatTraffic(traffic_day);
        int progress = 0;
        int layoutId = R.layout.notify_traffic_green;
        float trafficM = traffic_day/1024.0f/1024.0f;
        if (trafficM > limit_day*2) {
            progress = (int) ((trafficM>limit_day*3)?100:(trafficM-limit_day*2)*100/limit_day);
            layoutId = R.layout.notify_traffic_red;
        } else if (trafficM > limit_day) {
            progress = (int) ((trafficM>limit_day*2)?100:(trafficM-limit_day)*100/limit_day);
            layoutId = R.layout.notify_traffic_yellow;
        } else {
            progress = (int) (trafficM*100/limit_day);
        }
        Log.d(TAG, "progress="+progress);
        RemoteViews notify_traffic = new RemoteViews(this.getPackageName(), layoutId);
        notify_traffic.setTextViewText(R.id.tv_flow, desc);
        notify_traffic.setProgressBar(R.id.pb_flow, 100, progress, false);
        Intent intent = new Intent(this, MainActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this,
                R.string.app_name, intent, PendingIntent.FLAG_UPDATE_CURRENT);
        Notification.Builder builder = new Notification.Builder(this);
        builder.setContentIntent(contentIntent).setContent(notify_traffic)
                .setTicker("手机安全助手运行中").setSmallIcon(R.drawable.ic_app);
        mNotify = builder.build();
        startForeground(9, mNotify);
    }

    private Notification mNotify;
    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mNotify != null) {
```

```
                stopForeground(true);
            }
        }

        @Override
        public IBinder onBind(Intent intent) {
            return null;
        }
    }
```

6.7 小　结

本章主要介绍了 App 开发的自定义控件相关知识，包括自定义视图的步骤（声明属性、构造对象、测量尺寸、绘制视图）、自定义简单动画（任务片段、下拉刷新动画、圆弧进度动画）、自定义对话框的操作（对话框、改进日期对话框、自定义多级对话框）、自定义通知栏的用法（通知推送、进度条、远程视图）、Service 组件的基本用法（生命周期、3 种启停方式、推送服务到前台）。最后设计了一个实战项目“手机安全助手”，在该项目的 App 编码中采用了本章介绍的大部分自定义控件知识，以及服务启停和推送到通知栏的处理。另外，还介绍了如何获取手机上的应用包信息。

通过本章的学习，读者应该能够掌握以下 4 种开发技能。

（1）学会自定义简单控件，包括静止的视图和简单的动画。

（2）学会自定义对话框，在页面的合适位置显示和控制对话框。

（3）学会自定义通知栏，包括自定义样式与自定义操作的处理。

（4）学会 Service 组件的用法，如启停服务、绑定/解绑服务、把服务推送到前台等。



组合控件

本章介绍 App 开发常用的一些组合控件，主要包括底部标签栏的实现和用法、顶部导航栏的用法、横幅轮播条的实现和用法、循环视图 3 种布局的用法等。最后结合本章所学的知识演示一个实战项目“仿淘宝主页”的设计与实现。

7.1 标 签 栏

本节介绍底部标签栏的实现与用法，首先说明如何自定义实现标签按钮，然后介绍标签栏的 3 种实现方式，即 TabActivity 方式、ActivityGroup 方式和 FragmentActivity 方式。

7.1.1 标签按钮

按钮控件种类繁多，有文本按钮 Button、图像按钮 ImageButton、单选按钮 RadioButton、复选按钮 CheckBox、开关按钮 Switch 等，可展现的形式有文本、图像、文本+图标，如此丰富的展现形式，已经能够满足大部分控制需求。但总有少数场合比较特殊，一般的按钮样式满足不了，比如图 7-1 所示的微信底部标签栏，一排有 4 个标签按钮，每个按钮的图标和文字都会随着选中操作而高亮显示。



图 7-1 微信的底部标签栏

这样的标签栏控件是各大主流 App 的标配，无论是淘宝、京东，还是微信、手机 QQ，首屏底部一律是清一色的标签栏，而且在选中标签按钮时经常文字、图标、背景一起高亮显示。像这种标签按钮，Android 似乎没有对应的专门控件，如果要自定义控件，就得设计一个布局容器，里面放入一个文本控件和图像控件，然后注册选中事件的监听器，一旦监听到选中事件，就高亮显示文字、图标与布局背景。

自定义控件固然是一个不错的思路，不过无须如此大动干戈。读者还记得第 2 章介绍开关按钮 Switch 时结合状态图形与复选框实现仿 iOS 开关按钮的例子吧，通过状态图形自动展示选中与未选中两种状态的图像在外观上就像一个新控件。标签控件也是如此，要想高亮显示背景,可通过给 background 属性设置状态图形；要想高亮显示图标，可通过给 drawableTop 属性设置状态图形；高亮显示文本也能通过给 textColor 属性设置状态图形实现。这个小技巧估计很多人都没用过，既然文字、图标、背景都可以通过 StateDrawable 控制是否高亮显示，接下来的事情就好办了，具体的实现步骤如下：

01 定义一个状态图形的 XML 描述文件，当状态为选中时展示高亮图形，代码如下：

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_selected="true" android:color="@color/tab_text_selected" />
    <item android:color="@color/tab_text_normal" />
</selector>
```

02 在布局文件中给 TextView 控件的 background、textColor、drawableTop 三个属性分别设置对应的状态图形，设置代码举例如下：

```
    <TextView
        android:id="@+id/tv_tab_button"
        android:layout_width="100dp"
```

```
            android:layout_height="60dp"
            android:padding="5dp"
            android:layout_gravity="center"
            android:gravity="center"
            android:background="@drawable/tab_bg_selector"
            android:text="点我"
            android:textSize="12sp"
            android:textColor="@drawable/tab_text_selector"
            android:drawableTop="@drawable/tab_first_selector" />
```

03 在代码中调用 TextView 对象的 setSelected(true)方法时，该控件的文字、图标、背景同时高亮显示；调用 setSelected(false)方法时，该控件的文字、图标、背景恢复原状。具体效果如图 7-2 和图 7-3 所示，图 7-2 所示为尚未选中时的截图，图 7-3 所示为选中时的截图。



图 7-2　未选中标签按钮的截图



图 7-3　选中标签按钮的截图

是不是很神奇？接下来我们把该控件的共同属性挑出来，因为底部标签栏有 4、5 个标签按钮，如果每个按钮节点都添加重复的属性，就太啰嗦了，所以把它们之间通用的属性挑出来，然后在 values/styles.xml 中定义名为 TabButton 的新风格，具体的定义代码如下：

```
<style name="TabButton">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">match_parent</item>
    <item name="android:padding">5dp</item>
    <item name="android:layout_gravity">center</item>
    <item name="android:gravity">center</item>
    <item name="android:background">@drawable/tab_bg_selector</item>
    <item name="android:textSize">12sp</item>
    <item name="android:textStyle">normal</item>
    <item name="android:textColor">@drawable/tab_text_selector</item>
</style>
```

接下来，布局文件只要给 TextView 节点添加一行 style="@style/TabButton"，即可完成标签按钮的声明。直接在 styles.xml 中定义风格，无须另外编写自定义控件的代码，这是自定义控件的另一种途径。

7.1.2　实现底部标签栏

有了单个标签按钮，还需要一个边框把这些按钮放进去，自动响应每个按钮的点击操作，才能形成一个真正可用的底部标签栏。由于点击标签切换页面时标签栏自身仍保持不动，因此

这种情况不宜直接采用通常的活动页面跳转，只能通过特定形式完成页面切换。

标签栏的页面切换主要有 3 种方式：基于 TabActivity 的标签栏、基于 ActivityGroup 的标签栏和基于 FragmentActivity 的标签栏，3 种方式各有千秋。

1. 基于 TabActivity 的标签栏

TabActivity 原本就是设计用来做标签页面的，并且提供了 TabHost 和 TabWidget 两个控件，只不过它们仅用于标签栏，所以无须深入了解，套用固定的框架就行。

下面是 TabActivity 方式的布局文件代码：

```
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginBottom="@dimen/tabbar_height" />

        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:visibility="gone" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="@dimen/tabbar_height"
            android:layout_alignParentBottom="true"
            android:gravity="bottom"
            android:orientation="horizontal" >

            <LinearLayout
                android:id="@+id/ll_first"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:orientation="vertical" >
```

```
                <TextView
                    style="@style/TabButton"
                    android:drawableTop="@drawable/tab_first_selector"
                    android:text="@string/menu_first" />
            </LinearLayout>

            <LinearLayout
                android:id="@+id/ll_second"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:orientation="vertical" >

                <TextView
                    style="@style/TabButton"
                    android:drawableTop="@drawable/tab_second_selector"
                    android:text="@string/menu_second" />
            </LinearLayout>

            <LinearLayout
                android:id="@+id/ll_third"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:orientation="vertical" >

                <TextView
                    style="@style/TabButton"
                    android:drawableTop="@drawable/tab_third_selector"
                    android:text="@string/menu_third" />
            </LinearLayout>
        </LinearLayout>
    </RelativeLayout>
</TabHost>
```

有了布局文件，再来看对应的 Activity 框架，下面是 TabActivity 的代码：

```
public class TabHostActivity extends TabActivity implements OnClickListener {
    private static final String TAG = "TabHostActivity";
    private Bundle mBundle = new Bundle();
    private TabHost tab_host;
    private LinearLayout ll_first, ll_second, ll_third;
    private String FIRST_TAG = "first";
```

```
    private String SECOND_TAG = "second";
    private String THIRD_TAG = "third";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tab_host);
        mBundle.putString("tag", TAG);
        ll_first = (LinearLayout) findViewById(R.id.ll_first);
        ll_second = (LinearLayout) findViewById(R.id.ll_second);
        ll_third = (LinearLayout) findViewById(R.id.ll_third);
        ll_first.setOnClickListener(this);
        ll_second.setOnClickListener(this);
        ll_third.setOnClickListener(this);
        tab_host = getTabHost();
        tab_host.addTab(getNewTab(FIRST_TAG, R.string.menu_first,
                R.drawable.tab_first_selector, TabFirstActivity.class));
        tab_host.addTab(getNewTab(SECOND_TAG, R.string.menu_second,
                R.drawable.tab_second_selector, TabSecondActivity.class));
        tab_host.addTab(getNewTab(THIRD_TAG, R.string.menu_third,
                R.drawable.tab_third_selector, TabThirdActivity.class));
        tab_host.setCurrentTabByTag(FIRST_TAG);
        changeContainerView(ll_first);
    }

    private TabHost.TabSpec getNewTab(String spec, int label, int icon, Class<?> cls) {
        Intent intent = new Intent(this, cls).putExtras(mBundle);
        return tab_host.newTabSpec(spec).setContent(intent)
                .setIndicator(getString(label), getResources().getDrawable(icon));
    }

    @Override
    public void onClick(View v) {
        if (v.getId()==R.id.ll_first || v.getId()==R.id.ll_second || v.getId()==R.id.ll_third) {
            changeContainerView(v);
        }
    }

    private void changeContainerView(View v) {
        ll_first.setSelected(false);
        ll_second.setSelected(false);
        ll_third.setSelected(false);
        v.setSelected(true);
```



```
            if (v == ll_first) {
                tab_host.setCurrentTabByTag(FIRST_TAG);
            } else if (v == ll_second) {
                tab_host.setCurrentTabByTag(SECOND_TAG);
            } else if (v == ll_third) {
                tab_host.setCurrentTabByTag(THIRD_TAG);
            }
        }
    }
```

该方式的核心是 getNewTab 函数，方法内部可设置标签按钮的文本、图标以及该标签对应的活动页面。当发生标签按钮的点击事件时，系统调用 TabHost 的 setCurrentTabByTag 方法定位具体的切换页面。

具体的标签页切换效果如图 7-4 和图 7-5 所示。其中，图 7-4 所示为点击“首页”标签按钮时的截图，图 7-5 所示为点击“分类”标签按钮时的截图。



图 7-4　点击“首页”标签按钮



图 7-5　点击“分类”标签按钮

2. 基于 ActivityGroup 的标签栏

顾名思义，ActivityGroup 就是 Activity 的组合，允许在内部开启活动页面。从这个意义上来说，ActivityGroup 与 Activity 的关系相当于 Activity 与 Fragment 的关系。使用 ActivityGroup 实现标签栏有固定的模板，下面是 ActivityGroup 方式的布局文件代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/ll_container"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="bottom|center"
        android:orientation="horizontal" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="@dimen/tabbar_height"
        android:orientation="horizontal" >
```

```
        <LinearLayout
            android:id="@+id/ll_first"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:orientation="vertical" >

            <TextView
                style="@style/TabButton"
                android:drawableTop="@drawable/tab_first_selector"
                android:text="@string/menu_first" />
        </LinearLayout>

        <LinearLayout
            android:id="@+id/ll_second"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:orientation="vertical" >

            <TextView
                style="@style/TabButton"
                android:drawableTop="@drawable/tab_second_selector"
                android:text="@string/menu_second" />
        </LinearLayout>

        <LinearLayout
            android:id="@+id/ll_third"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:orientation="vertical" >

            <TextView
                style="@style/TabButton"
                android:drawableTop="@drawable/tab_third_selector"
                android:text="@string/menu_third" />
        </LinearLayout>
    </LinearLayout>
</LinearLayout>
```

与上面的布局文件对应的是 ActivityGroup 的代码：

```
public class TabGroupActivity extends ActivityGroup implements OnClickListener {
    private static final String TAG = "TabGroupActivity";
    private Bundle mBundle = new Bundle();
    private LinearLayout ll_container, ll_first, ll_second, ll_third;
```



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_tab_group);
    ll_container = (LinearLayout) findViewById(R.id.ll_container);
    ll_first = (LinearLayout) findViewById(R.id.ll_first);
    ll_second = (LinearLayout) findViewById(R.id.ll_second);
    ll_third = (LinearLayout) findViewById(R.id.ll_third);
    ll_first.setOnClickListener(this);
    ll_second.setOnClickListener(this);
    ll_third.setOnClickListener(this);
    mBundle.putString("tag", TAG);
    changeContainerView(ll_first);
}

@Override
public void onClick(View v) {
    if (v.getId()==R.id.ll_first || v.getId()==R.id.ll_second || v.getId()==R.id.ll_third) {
        changeContainerView(v);
    }
}

private void changeContainerView(View v) {
    ll_first.setSelected(false);
    ll_second.setSelected(false);
    ll_third.setSelected(false);
    v.setSelected(true);
    if (v == ll_first) {
        toActivity("first", TabFirstActivity.class);
    } else if (v == ll_second) {
        toActivity("second", TabSecondActivity.class);
    } else if (v == ll_third) {
        toActivity("third", TabThirdActivity.class);
    }
}

private void toActivity(String label, Class<?> cls) {
    Intent intent = new Intent(this, cls).putExtras(mBundle);
    ll_container.removeAllViews();
    View v = getLocalActivityManager().startActivity(label, intent).getDecorView();
    v.setLayoutParams(new LayoutParams(
            LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
```

```
            ll_container.addView(v);
        }
    }
```

该方式的核心是 toActivity 函数，方法内部可设置标签按钮的文本、图标以及该标签对应的活动页面。从函数中可以看到，startActivity 方法返回一个 Window 对象，然后从该 Window 对象提取标签页的实际视图（调用 getDecorView 方法）。我们可以把 DecorView 理解为该标签页的根视图，将这个根视图 DecorView 加入 ActivityGroup 的视图容器中。注意，这里在调用 startActivity 方法前需要先调用 getLocalActivityManager 方法获得页面管理器，才能进行后续操作，getLocalActivityManager 方法是 ActivityGroup 特有的函数。

该方式的标签栏页面效果与 TabActivity 一样。为了区分两种方式，这里在具体标签页中把来源打印出来，如图 7-6 和图 7-7 所示。其中，图 7-6 所示为点击“首页”标签按钮时的截图，图 7-7 所示为点击“购物车”标签按钮时的截图。



图 7-6　点击“首页”标签按钮

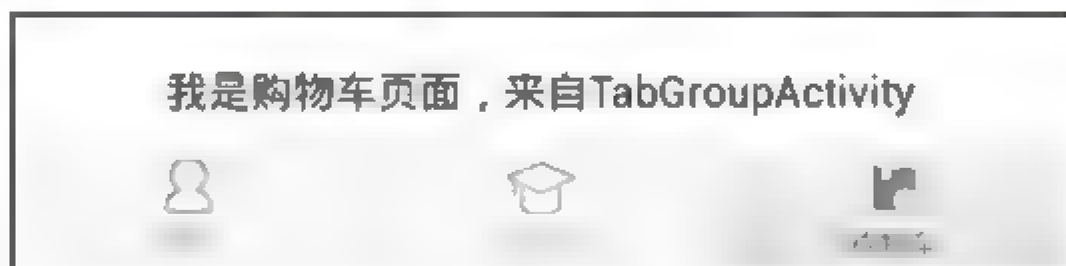


图 7-7　点击“购物车”标签按钮

3. 基于 FragmentActivity 的标签栏

前面提到，ActivityGroup 方式采用一个 ActivityGroup 对应多个 Activity 的做法，那么也可以采取一个 Activity 对应多个 Fragment 的做法，基于 FragmentActivity 的标签栏就是该思路的第 3 种方式。与前两种方式一样，FragmentActivity 也有固定的使用模板，下面是该方式的布局文件代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <!-- 把 FragmentLayout 放在 FragmentTabHost 上面，标签页就在页面底部；
        反之 FragmentLayout 在 FragmentTabHost 下面，标签页就在页面顶部。 -->
    <FrameLayout
        android:id="@+id/realtabcontent"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />

    <android.support.v4.app.FragmentTabHost
        android:id="@android:id/tabhost"
        android:layout_width="match_parent"
        android:layout_height="@dimen/tabbar_height" >
```

```
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="0dp"
            android:layout_height="0dp"
            android:layout_weight="0" />

    </android.support.v4.app.FragmentTabHost>
</LinearLayout>
```

看起来布局文件简洁了许多，该方式的代码也同样简洁了：

```
public class TabFragmentActivity extends FragmentActivity {
    private static final String TAG = "TabFragmentActivity";
    private FragmentTabHost mTabHost;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tab_fragment);
        Bundle bundle = new Bundle();
        bundle.putString("tag", TAG);
        mTabHost = (FragmentTabHost) findViewById(android.R.id.tabhost);
        mTabHost.setup(this, getSupportFragmentManager(), R.id.realtabcontent);
        //addTab(标题，跳转的 Fragment，传递参数的 Bundle)
        mTabHost.addTab(getTabView(R.string.menu_first, R.drawable.tab_first_selector),
                TabFirstFragment.class, bundle);
        mTabHost.addTab(getTabView(R.string.menu_second, R.drawable.tab_second_selector),
                TabSecondFragment.class, bundle);
        mTabHost.addTab(getTabView(R.string.menu_third, R.drawable.tab_third_selector),
                TabThirdFragment.class, bundle);
        //设置 tabs 之间的分隔线不显示
        mTabHost.getTabWidget().setShowDividers(LinearLayout.SHOW_DIVIDER_NONE);
    }

    private TabSpec getTabView(int textId, int imgId) {
        String text = getResources().getString(textId);
        Drawable drawable = getResources().getDrawable(imgId);
        //必须设置图片大小，否则不显示
        drawable.setBounds(0, 0, drawable.getMinimumWidth(), drawable.getMinimumHeight());
        //R.layout.item_tabbar 是单个标签按钮的布局文件
        View item_tabbar = getLayoutInflater().inflate(R.layout.item_tabbar, null);
        TextView tv_item = (TextView) item_tabbar.findViewById(R.id.tv_item_tabbar);
        tv_item.setText(text);
        tv_item.setCompoundDrawables(null, drawable, null, null);
```

```
            TabSpec spec = mTabHost.newTabSpec(text).setIndicator(item_tabbar);
            return spec;
        }
    }
```

FragmentActivity 方式的核心是 addTab 函数，内部可自定义每个标签按钮的视图和对应的 Fragment 页面。因为 FragmentTabHost 已经自动处理了点击事件，所以无须另外调用 setSelected 方法。该方式与前两种方式的不同之处在于标签页是 Fragment 而不是 Activity，因此标签页内部无法直接操作选项菜单。

FragmentActivity 方式的标签栏与前两种方式在形式上没什么差别，具体效果如图 7-8 和图 7-9 所示。其中，图 7-8 所示为点击“分类”标签按钮时的截图，图 7-9 所示为点击“购物车”标签按钮时的截图。



图 7-8　点击“分类”标签按钮



图 7-9　点击“购物车”标签按钮

7.2　导　航　栏

本节介绍导航栏的组成控件，包括工具栏 Toolbar、溢出菜单 OverflowMenu、搜索框 SearchView、标签布局 TabLayout 的相关用法，以及如何定制 Toolbar 的视图与 TabLayout 的标签页。

7.2.1　工具栏 Toolbar

主流 App 除了底部有一排标签栏外，通常顶部还有一排导航栏。在 Android5.0 之前，这个顶部导航栏以 ActionBar 控件的形式出现，但 ActionBar 存在不灵活、难以扩展等毛病，所以 Android5.0 之后推出了 Toolbar 工具栏控件，意在取代 ActionBar。

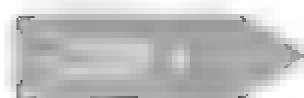
不过为了兼容之前的版本，ActionBar 控件仍然保留。Toolbar 与 ActionBar 都占着顶部导航栏的位置，要想引入 Toolbar 就得先关闭 ActionBar。具体的操作步骤如下：

01 在 styles.xml 中定义一个不包含 ActionBar 的风格样式，代码如下：

```
<style name="AppCompatTheme" parent="Theme.AppCompat.Light.NoActionBar" />
```

02 修改 AndroidManifest.xml，把 activity 节点的 android:theme 属性值改为第一步定义的风格，如 android:theme="@style/AppCompatTheme"。

03 将页面布局文件的根节点改为 LinearLayout，且为 vertical 垂直方向；然后增加一个 Toolbar 元素，因为 Toolbar 本质是一个 ViewGroup，所以也可以在下面添加别的控件。下面是一个布局文件的片段：



```
<android.support.v7.widget.Toolbar
    android:id="@+id/tl_head"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

步骤 04 将 Activity 代码改为继承自 AppCompatActivity，其实在 Android Studio 中新建模块已经是默认继承 AppCompatActivity 了。然后在 onCreate 函数中获取布局文件中的 Toolbar 对象，并调用 setSupportActionBar 方法设置当前的 Toolbar 对象。

Toolbar 之所以比 ActionBar 灵活，原因是 Toolbar 提供了多个属性指定控件风格。Toolbar 的常用属性及设置方法见表 7-1（自定义属性的用法参见第 6 章的“6.1.1 声明属性”）。

表 7-1 Toolbar 的常用属性及设置方法说明

XML 中的属性	Toolbar 类的设置方法	说明
logo	setLogo	设置工具栏图标
title	setTitle	设置标题文字
titleTextColor	setTitleTextColor	设置标题的文字颜色
titleTextAppearance	setTitleTextAppearance	设置标题的文字风格。风格定义在 styles.xml 中
subtitle	setSubtitle	设置副标题文字。副标题在标题下方
subtitleTextColor	setSubtitleTextColor	设置副标题的文字颜色
subtitleTextAppearance	setSubtitleTextAppearance	设置副标题的文字风格
navigationIcon	setNavigationIcon	设置左侧导航图标
无	setNavigationOnClickListener	设置导航图标的点击监听器

下面是使用 Toolbar 的代码片段：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_toolbar);
    Toolbar tl_head = (Toolbar) findViewById(R.id.tl_head);
    tl_head.setNavigationIcon(R.drawable.ic_back);
    tl_head.setTitle("工具栏页面");
    tl_head.setTitleTextColor(Color.RED);
    tl_head.setLogo(R.drawable.ic_app);
    tl_head.setSubtitle("Toolbar");
    tl_head.setSubtitleTextColor(Color.YELLOW);
    tl_head.setBackgroundResource(R.color.blue_light);
    setSupportActionBar(tl_head);
    //setNavigationOnClickListener 必须放到 setSupportActionBar 之后，不然不起作用
    tl_head.setNavigationOnClickListener(new OnClickListener() {
        @Override
```

```
                public void onClick(View view) {
                    finish();
                }
            });
        }
```

具体的工具栏效果如图 7-10 所示，该工具栏的界面元素包括导航图标、工具栏图标、标题、副标题。



图 7-10　简单设置后的工具栏界面

7.2.2　溢出菜单 OverflowMenu

导航栏右边往往有个三点图标，点击后会弹出菜单。这个右上角的弹出菜单名叫溢出菜单 OverflowMenu，意指导航栏不够放了、溢出来了。溢出菜单其实就是把选项菜单 OptionsMenu 搬到了页面右上方，具体的菜单布局与代码用法基本同选项菜单，不同之处在于溢出菜单多了个 showAsAction 属性，该属性用来控制菜单项在导航栏上的展示位置，具体的取值说明见表 7-2。

表 7-2　菜单项展示位置类型的取值说明

展示位置类型	说明
always	总是在导航栏上显示菜单图标
ifRoom	如果导航栏右侧有空间，该项就直接显示在导航栏上，不再放入溢出菜单
never	从不在导航栏上直接显示，一直放在溢出菜单列表里面
withText	如果能在导航栏上显示，除了显示图标，还要显示该项的文字说明
collapseActionView	操作视图要折叠为一个按钮，点击该按钮再展开操作视图，主要用于 SearchView

默认情况下，菜单列表的菜单项不会在文字左边显示图标，即使在菜单布局中设置了 icon 属性也没有作用。所以想让菜单项显示左侧图标就得调用 MenuBuilder 的 setOptionalIconsVisible 方法。该方法是一个隐藏方法，只能通过反射机制调用。具体的调用代码如下：

```
    public static void setOverflowIconVisible(int featureId, Menu menu) {
        // ActionBar 的 featureId 是 8，Toolbar 的 featureId 是 108
        if (featureId % 100 == Window.FEATURE_ACTION_BAR && menu != null) {
            if (menu.getClass().getSimpleName().equals("MenuBuilder")) {
                try {
                    Method m = menu.getClass().getDeclaredMethod(
                            "setOptionalIconsVisible", Boolean.TYPE);
                    m.setAccessible(true);
                    m.invoke(menu, true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    }
```



另外，菜单布局中将 showAsAction 属性设置为 ifRoom 或 always，不过即使工具栏上还有空间，该菜单项也不会显示在工具栏上。这方面也很不好，因为在 ActionBar 时代，这么做没问题，到 Toolbar 时代反而出了问题。既然有问题就得解决，解决办法挺简单，首先在菜单布局的 menu 根节点增加命名空间声明 xmlns:app="http://schemas.android.com/apk/res-auto"，然后把 android:showAsAction="ifRoom"改为 app:showAsAction="ifRoom"。很眼熟是不是？这分明就是自定义属性的做法。下面来看用于溢出菜单的布局文件代码：

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" >

    <item
        android:id="@+id/menu_refresh"
        android:orderInCategory="1"
        android:icon="@drawable/ic_refresh"
        app:showAsAction="ifRoom"
        android:title="刷新"/>

    <item
        android:id="@+id/menu_about"
        android:orderInCategory="8"
        android:icon="@drawable/ic_about"
        app:showAsAction="never"
        android:title="关于"/>

    <item
        android:id="@+id/menu_quit"
        android:orderInCategory="9"
        android:icon="@drawable/ic_quit"
        app:showAsAction="never"
        android:title="退出"/>
</menu>
```

下面是在页面代码中操作溢出菜单的代码片段：

```
    @Override
    public boolean onMenuOpened(int featureId, Menu menu) {
        Utils.setOverflowIconVisible(featureId, menu);   // 显示菜单项左侧的图标
        return super.onMenuOpened(featureId, menu);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_overflow, menu);
        return true;
    }
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();
    if (id == android.R.id.home) {   //该分支响应导航图标的点击动作
        finish();
    } else if (id == R.id.menu_refresh) {
        tv_desc.setText("当前刷新时间："+Utils.getNowDateTime("yyyy-MM-dd HH:mm:ss"));
        return true;
    } else if (id == R.id.menu_about) {
        Toast.makeText(this, "这个是工具栏的演示 demo", Toast.LENGTH_LONG).show();
        return true;
    } else if (id == R.id.menu_quit) {
        finish();
    }
    return super.onOptionsItemSelected(item);
}
```

添加溢出菜单后的导航栏效果如图 7-11 和图 7-12 所示。其中，图 7-11 所示为导航栏的初始界面，此时导航栏右侧有一个刷新按钮，还有一个三点图标；点击三点图标，弹出剩余的菜单项列表，如图 7-12 所示。



图 7-11　溢出菜单初始界面



图 7-12　点击按钮弹出菜单列表

7.2.3　搜索框 SearchView

导航栏中间往往有个搜索框，特别是电商 App 的导航栏，搜索框是标配。在工具栏上添加并使用搜索框有些复杂，实现步骤大致如下：

步骤 01 在菜单布局文件中定义搜索项，示例代码如下：

```
<item
    android:id="@+id/menu_search"
    android:orderInCategory="1"
    android:icon="@drawable/ic_search"
    app:showAsAction="ifRoom"
    android:title="搜索"
    app:actionViewClass="android.support.v7.widget.SearchView" />
```

步骤 02 在 res\xml 目录下新建 searchable.xml，设置搜索框的样式代码，举例如下：

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/app_name"
```

```
android:hint="@string/please_input"
android:inputType="text"
android:searchButtonText="@string/search" />
```

03 在 AndroidManifest.xml 中加入一个搜索结果页面的 activity 节点定义，需要指定 action 和 meta-data，举例如下：

```
<activity android:name=".SearchResultActvity" android:theme="@style/AppCompatTheme" >
    <intent-filter>
        <action android:name="android.intent.action.SEARCH"/>
    </intent-filter>
    <meta-data android:name="android.app.searchable" android:resource="@xml/searchable"/>
</activity>
```

04 在 Activity 代码中初始化搜索框，并关联搜索动作对应的结果 Activity，如 SearchResultActvity。代码片段如下：

```
private void initSearchView(Menu menu) {
    MenuItem menuItem = menu.findItem(R.id.menu_search);
    SearchView searchView = (SearchView) MenuItemCompat.getActionView(menuItem);
    if (searchView == null) {
        Log.d(TAG, "Fail to get SearchView.");
    } else {
        if (getIntent() != null) {
            //设置是否将搜索视图默认折叠为图标
            searchView.setIconifiedByDefault(getIntent().getBooleanExtra("collapse", true));
        } else {
            searchView.setIconifiedByDefault(true);
        }
        //设置是否启用完成图标
        searchView.setSubmitButtonEnabled(true);
        SearchManager searchManager = (SearchManager) getSystemService(Context.SEARCH_
SERVICE);
        ComponentName cn = new ComponentName(this, SearchResultActvity.class);
        SearchableInfo info = searchManager.getSearchableInfo(cn);
        if (info == null) {
            Log.d(TAG, "Fail to get SearchResultActvity.");
        }
        //设置搜索动作的定义
        searchView.setSearchableInfo(info);
        sac_key = (SearchView.SearchAutoComplete) searchView.findViewById(R.id.search_src_text);
        sac_key.setTextColor(Color.WHITE);
        sac_key.setHintTextColor(Color.WHITE);
        //设置搜索关键字的监听器，如是否展示关键字的匹配列表
        searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
```

```
                @Override
                public boolean onQueryTextSubmit(String query) {
                    return false;
                }

                @Override
                public boolean onQueryTextChange(String newText) {
                    doSearch(newText);
                    return true;
                }
            });
            Bundle bundle = new Bundle();
            bundle.putString("hi", "hello");
            searchView.setAppSearchData(bundle);
        }
    }

    private SearchView.SearchAutoComplete sac_key;
    private String[] hintArray = { "iphone", "iphone7s", "iphone7", "iphone7 plus", "iphone6s", "iphone6",
"iphone6 plus"};
    private void doSearch(String text) {
        if (text.indexOf("i") == 0) {
            ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                    R.layout.search_list_auto, hintArray);
            sac_key.setAdapter(adapter);
            sac_key.setOnItemClickListener(new OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                    TextView tv_item = (TextView) view;
                    sac_key.setText(tv_item.getText());
                }
            });
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_search, menu);
        //对搜索框做初始化
        initSearchView(menu);
        return true;
    }
```

05 编写搜索结果页面的 Activity 代码，获取关键字的代码片段如下：

```
private void doSearchQuery(Intent intent) {
    if (intent == null) {
        return;
    } else {
        // 如果通过 ACTION_SEARCH 调用，即通过搜索调用
        if (Intent.ACTION_SEARCH.equals(intent.getAction())) {
            // 获取额外信息
            Bundle bundle = intent.getBundleExtra(SearchManager.APP_DATA);
            String value = bundle.getString("hi");
            // 获取搜索内容
            String queryString = intent.getStringExtra(SearchManager.QUERY);
            tv_search_result.setText("您输入的搜索文字是："+queryString+", 额外信息："+value);
        }
    }
}
```

搜索框的使用效果如图 7-13～图 7-16 所示。其中，图 7-13 所示为导航栏的初始界面；图 7-14 为点击搜索图标后，展开搜索视图的界面；图 7-15 所示为输入搜索文字后，弹出关键词列表的界面；图 7-16 所示为点击完成按钮，跳转到搜索结果页面的截图。



图 7-13 搜索框初始页面



图 7-14 展开搜索框的页面

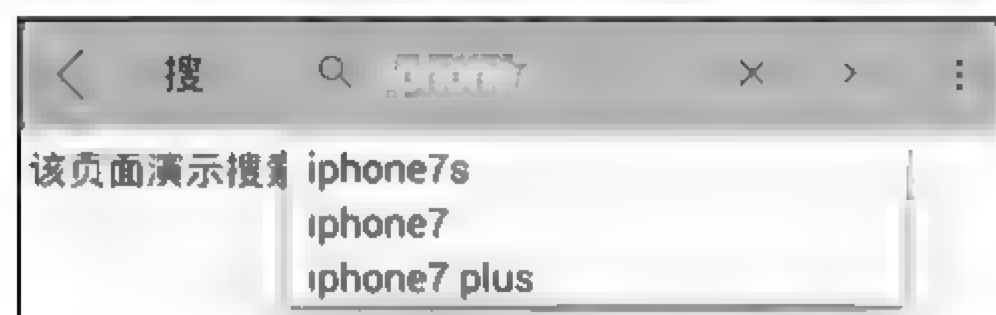


图 7-15 输入关键字弹出选择列表



图 7-16 搜索结果页面的截图

7.2.4 标签布局 TabLayout

Toolbar 作为 ActionBar 的升级版，好处在于允许设置内部控件的样式，还允许添加其他外部控件。第 6 章的实战项目"手机安全助手"流量主页面的顶部是一个自己做的简单导航栏，该导航栏的主节点是 LinearLayout，现在我们把 LinearLayout 换成 Toolbar，相当于系统默认实现左侧的导航图标和右侧的溢出菜单，中间的部分是开发者要添加的视图。

下面是修改后的布局文件片段，此时 Toolbar 节点可以当作 LinearLayout 节点使用：

```
<android.support.v7.widget.Toolbar
    android:id="@+id/tl_head"
    android:layout_width="match_parent"
```

```
    android:layout_height="50dp"
    android:background="@color/blue_light"
    app:navigationIcon="@drawable/ic_back" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/tv_day"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_centerInParent="true"
            android:background="@drawable/editext_selector"
            android:gravity="center"
            android:textColor="@color/black"
            android:textSize="17sp" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_toLeftOf="@+id/tv_day"
            android:gravity="center"
            android:text="统计日期 "
            android:textColor="@color/black"
            android:textSize="17sp" />
    </RelativeLayout>
</android.support.v7.widget.Toolbar>
```

修改后的导航栏效果如图 7-17 所示，中部原来展示标题的位置变成展示统计日期了。

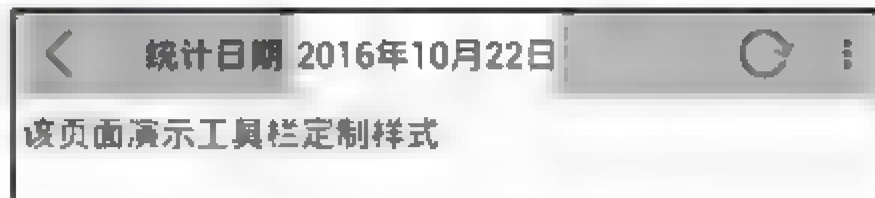


图 7-17　定制修改后的导航栏

如果定制 Toolbar 仅仅放入几个基本控件，就太小儿科了，这么好的工具栏，必须有杀手级别的控件搭配。下面先看京东 App 的两张截图，图 7-18 是商品页面，图 7-19 是详情页面，这两个页面之间通过左右滑动切换。导航栏上有文字标签，类似于翻页标题栏 PagerTabStrip，用于指示当前滑到了哪个页面。

通过工具栏控制页面左右滑动的用户体验挺不错，这里压轴用的便是 design 库中的标签布局 TabLayout，使用该控件前要先修改 build.gradle，在 dependencies 节点中加入一行代码表示导入 design 库：

```
compile 'com.android.support:design:25.1.0'
```



图 7-18　京东的商品页面截图



图 7-19　京东的详情页面截图

TabLayout 的展现形式类似于 PagerTabStrip，同样是文字标签带下划线，不同的是 TabLayout 允许定制更丰富的样式，新增的样式属性主要有以下 6 种。

- tabBackground：指定标签的背景。
- tabIndicatorColor：指定下划线的颜色。
- tabIndicatorHeight：指定下划线的高度。
- tabTextColor：指定标签文字的颜色。
- tabTextAppearance：指定标签文字的风格。
- tabSelectedTextColor：指定选中文字的颜色。

下面是在 XML 文件中使用 TabLayout 的布局代码片段：

```
<android.support.v7.widget.Toolbar
    android:id="@+id/tl_head"
    android:layout_width="match_parent"
    android:layout_height="50dp"
    app:navigationIcon="@drawable/ic_back" >

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <android.support.design.widget.TabLayout
            android:id="@+id/tab_title"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_centerInParent="true"
            app:tabIndicatorColor="@color/red"
            app:tabIndicatorHeight="2dp"
            app:tabSelectedTextColor="@color/red"
```

```
                    app:tabTextColor="@color/grey"
                    app:tabTextAppearance="@style/TabText" />

        </RelativeLayout>
    </android.support.v7.widget.Toolbar>
```

在代码中，TabLayout 通过以下 4 种方法操作标签。

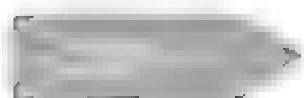
- newTab：创建新标签。
- addTab：添加一个标签。
- getTabAt：获取指定位置的标签。
- setOnTabSelectedListener：设置标签的选中监听器。该监听器需实现 OnTabSelectedListener 接口的 3 个方法。
  - onTabSelected：标签被选中时触发。
  - onTabUnselected：标签被取消选中时触发。
  - onTabReselected：标签被重新选中时触发。

把 TabLayout 与 ViewPager 结合起来就是一个固定的套路，使用时直接套框架就行。下面是两者联合使用的代码：

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tab_layout);
        tl_head = (Toolbar) findViewById(R.id.tl_head);
        tab_title = (TabLayout) findViewById(R.id.tab_title);
        vp_content = (ViewPager) findViewById(R.id.vp_content);
        setSupportActionBar(tl_head);
        mTitleArray.add("商品");
        mTitleArray.add("详情");
        initTabLayout();
        initTabViewPager();
    }

    private void initTabLayout() {
        tab_title.addTab(tab_title.newTab().setText(mTitleArray.get(0)));
        tab_title.addTab(tab_title.newTab().setText(mTitleArray.get(1)));
        tab_title.setOnTabSelectedListener(this);
    }

    private void initTabViewPager() {
        GoodsPagerAdapter adapter = new GoodsPagerAdapter(getSupportFragmentManager(),
mTitleArray);
        vp_content.setAdapter(adapter);
```



```
        vp_content.addOnPageChangeListener(new SimpleOnPageChangeListener() {
            @Override
            public void onPageSelected(int position) {
                tab_title.getTabAt(position).select();
            }
        });
    }

    @Override
    public void onTabReselected(Tab tab) {
    }

    @Override
    public void onTabSelected(Tab tab) {
        vp_content.setCurrentItem(tab.getPosition());
    }

    @Override
    public void onTabUnselected(Tab tab) {
    }
```

接下来看在工具栏上显示标签页的效果。选中“商品”标签，页面下方显示商品信息文字，如图 7-20 所示；然后选中“详情”标签，切换到商品详情页面，如图 7-21 所示。感觉不错吧，赶快动手实践一下，你也可以实现京东 App 的标签导航栏。



图 7-20　点击“商品”标签

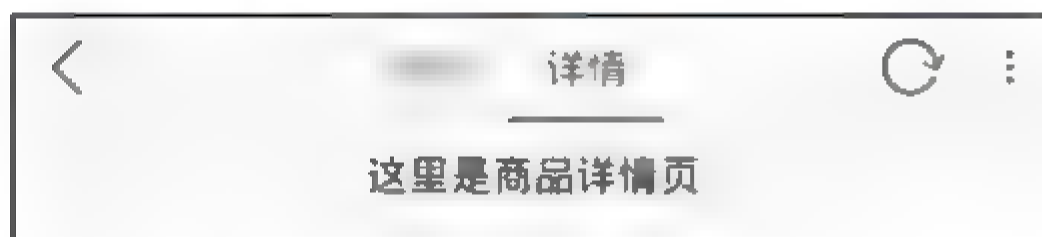


图 7-21　点击“详情”标签

TabLayout 默认采用文本标签，也支持自定义标签，除了放文本还可以放图像，比如加一个角标。自定义标签的过程很简单，首先要定义标签项的布局文件。下面是一个布局文件的例子，其中包含文本控件与图像控件，并且 TextView 的 textColor 属性与 ImageView 的 src 属性都采用状态图形，代码如下：

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/tv_toolbar1"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_centerInParent="true"
```

```
            android:gravity="center"
            android:textColor="@drawable/toolbar_text_selector"
            android:textSize="17sp" />

        <ImageView
            android:id="@+id/iv_point1"
            android:layout_width="25dp"
            android:layout_height="25dp"
            android:layout_toRightOf="@+id/tv_toolbar1"
            android:paddingTop="10dp"
            android:paddingLeft="3dp"
            android:scaleType="fitCenter"
            android:src="@drawable/toolbar_image_selector" />"
</RelativeLayout>
```

然后打开活动页面代码，只要修改 initTabLayout 函数即可，关键是调用了 setCustomView 方法，代码如下：

```
    private void initTabLayout() {
        tab_title.addTab(tab_title.newTab().setCustomView(R.layout.item_toolbar1));
        tv_toolbar1 = (TextView) findViewById(R.id.tv_toolbar1);
        tv_toolbar1.setText(mTitleArray.get(0));
        tab_title.addTab(tab_title.newTab().setCustomView(R.layout.item_toolbar2));
        tv_toolbar2 = (TextView) findViewById(R.id.tv_toolbar2);
        tv_toolbar2.setText(mTitleArray.get(1));
        tab_title.setOnTabSelectedListener(new ViewPagerOnTabSelectedListener(vp_content));
    }
```

重新编译并运行 App，最新的效果如图 7-22 和图 7-23 所示。其中，图 7-22 所示为点击“商品”标签时的界面，此时“商品”文字右上角显示红点；图 7-23 所示为点击“详情”标签时的界面，此时“详情”文字右上角显示红点。



图 7-22　点击“商品”的自定义标签



图 7-23　点击“详情”的自定义标签

7.3 横　幅　条

本节介绍横幅条 Banner 的两种展现形式与具体实现，包括如何在 Banner 底部自定义可以滚动的指示器、如何实现会自动轮播的横幅条。同时还会复习自定义视图和自定义动画的知识。



7.3.1　自定义指示器

在第 5 章介绍 ViewPager 时给出了启动引导页的例子，为了让用户知道当前是在第几页，在每个页面下方都要添加一排圆点，通过高亮圆点指示当前的页面位置，这排圆点我们称之为指示器。引导页里的指示器其实附着在每个 Fragment 页面下方，而不是固定在手机屏幕下方，所以会感觉有些奇怪。理想的情况是，引导页在滑动时屏幕下方的指示器固定不动，高亮圆点随着页面滑动而缓慢挪动，页面滑到下一页，高亮圆点刚好挪到下一个圆点处。

这么说可能有些抽象，不如看看新方式的效果图，如图 7-24 所示。当前翻页位置在第一页和第二页之前，此时底部指示器的高亮圆点刚好挪到第一个圆点与第二个圆点之间，随着页面的滚动，高亮圆点随之平滑滚动。

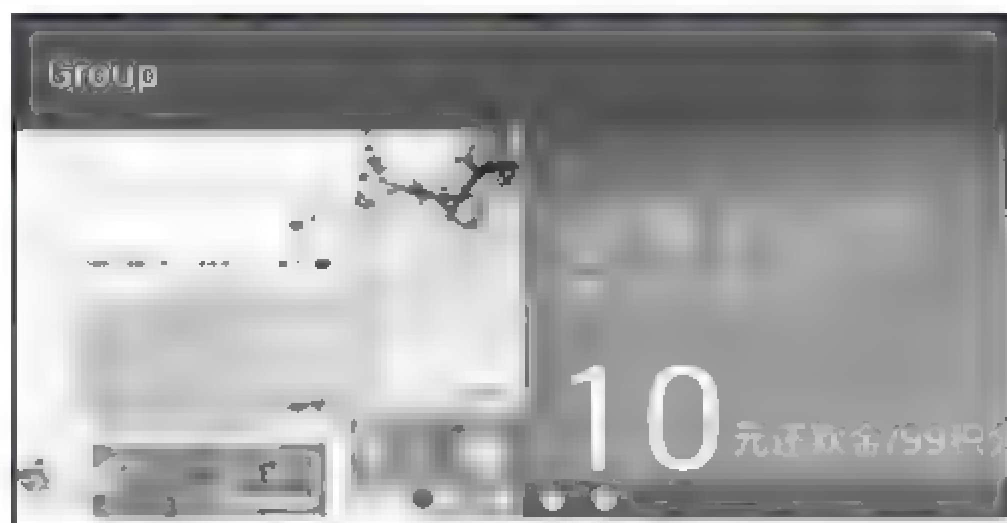


图 7-24　底部滑动着的高亮圆点

要实现指示器的平滑滚动效果，得用到 ViewPager 的页面变化监听器 OnPageChangeListener。第 5 章介绍该监听器时提到有 onPageScrollStateChanged、onPageScrolled、onPageSelected 三个方法，在具体场合有下面两种用法。

1. 只实现 onPageSelected 方法，在页面滚动结束时触发，该用法是最常见的。

在这种情况下，onPageScrollStateChanged 和 onPageScrolled 两个方法成了摆设，占着多余的代码行非常浪费。此时不必完整实现 OnPageChangeListener 接口，只需创建一个 SimpleOnPageChangeListener 实例即可，该内部类在 ViewPager 源码中已经封装好了，开发者只要实现 onPageSelected 方法就行。具体的调用代码如下：

```
mPager.addOnPageChangeListener(new SimpleOnPageChangeListener() {
    @Override
    public void onPageSelected(int position) {
        setButton(position);
    }
});
```

2. 除了实现 onPageSelected 方法，还要实现 onPageScrollStateChanged 和 onPageScrolled 两个方法。

这种情况适用于指示器，特别是 onPageScrolled 方法的参数已明确指出当前的滚动进度，正好给指示器的滚动位置提供参考。接下来的工作是自定义一个指示器控件，首先绘制背景图

的一排圆点，然后绘制前景图的高亮圆点。正好复习一下第 6 章自定义视图的技术，读者可自定义实现该指示器控件，下面是该控件的参考代码：

```
public class PagerIndicator extends LinearLayout {
    private Context mContext;
    private int mCount = 5;
    private int mPad = 30;
    private int mSeq = 0;
    private float mRatio = 0.0f;
    private Paint mPaint;
    private Bitmap mBackImage;
    private Bitmap mForeImage;

    public PagerIndicator(Context context) {
        this(context, null);
    }

    public PagerIndicator(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        init();
    }

    private void init() {
        mPaint = new Paint();
        mBackImage = BitmapFactory.decodeResource(getResources(), R.drawable.icon_point_n);
        mForeImage = BitmapFactory.decodeResource(getResources(), R.drawable.icon_point_c);
    }

    @Override
    protected void dispatchDraw(Canvas canvas) {
        super.dispatchDraw(canvas);
        int left = (getMeasuredWidth() - mCount*mPad) / 2;
        for (int i=0; i<mCount; i++) {
            canvas.drawBitmap(mBackImage, left+i*mPad, 0, mPaint);
        }
        canvas.drawBitmap(mForeImage, left+(mSeq+mRatio)*mPad, 0, mPaint);
    }

    public void setCount(int count, int pad) {
        mCount = count;
        mPad = pad;
        invalidate();
```

```
    }

    public void setCurrent(int seq, float ratio) {
        mSeq = seq;
        mRatio = ratio;
        invalidate();
    }
}
```

有了自定义的指示器控件，就可以重写 OnPageChangeListener 接口的 onPageScrolled 方法了。在该方法中调用指示器的 setCurrent 方法就能动态刷新高亮圆点的滚动动画，滚动效果如图 7-24 所示。具体的调用代码举例如下：

```
    private class BannerChangeListener implements ViewPager.OnPageChangeListener {
        @Override
        public void onPageScrollStateChanged(int arg0) {
        }

        @Override
        public void onPageScrolled(int seq, float ratio, int offset) {
            mIndicator.setCurrent(seq, ratio);
        }

        @Override
        public void onPageSelected(int seq) {
            mIndicator.setCurrent(seq, 0);
        }
    }
```

7.3.2 实现横幅轮播 Banner

前面给 ViewPager 加了指示器，不过仍然是静止页面，只有用户在屏幕上左右滑动时才会进行翻页动作。看看电商 App 的首页，显眼位置的 Banner 会自动滚动，每隔两三秒就轮播下一个广告页，让页面熠熠生辉。不过这难不倒我们，自动滚动不就是加一个动画效果么？第 6 章的自定义动画知识正好派上用场。只要结合 Handler+Runnable，实现一个简单动画非常容易。下面是自定义 Banner 的代码，相当于启动引导页的代码加上 Handler 与 Runnable 组合：

```
public class BannerPager extends RelativeLayout implements View.OnClickListener {
    private static final String TAG = "BannerPager";
    private Context mContext;
    private ViewPager mPager;
    private List<ImageView> mViewList = new ArrayList<ImageView>();
    private RadioGroup mGroup;
    private int mCount;
    private LayoutInflater mInflater;
```

```
    private int dip_15;
    private static int mInterval = 2000;

    public BannerPager(Context context) {
        this(context, null);
    }

    public BannerPager(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        init();
    }

    public void start() {
        mHandler.postDelayed(mScroll, mInterval);
    }

    public void stop() {
        mHandler.removeCallbacks(mScroll);
    }

    public void setInterval(int interval) {
        mInterval = interval;
    }

    public void setImage(ArrayList<Integer> imageList) {
        for (int i = 0; i < imageList.size(); i++) {
            Integer imageID = ((Integer) imageList.get(i)).intValue();
            ImageView iv = new ImageView(mContext);
            iv.setLayoutParams(new LayoutParams(
                    LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
            iv.setScaleType(ImageView.ScaleType.FIT_XY);
            iv.setImageResource(imageID);
            iv.setOnClickListener(this);
            mViewList.add(iv);
        }
        mPager.setAdapter(new ImageAdapater());
        mPager.addOnPageChangeListener(new SimpleOnPageChangeListener() {
            @Override
            public void onPageSelected(int position) {
                setButton(position);
            }
        });
```

```
        mCount = imageList.size();
        for (int i = 0; i < mCount; i++) {
            RadioButton radio = new RadioButton(mContext);
            radio.setLayoutParams(new RadioGroup.LayoutParams(dip_15, dip_15));
            radio.setGravity(Gravity.CENTER);
            radio.setButtonDrawable(R.drawable.indicator_selector);
            mGroup.addView(radio);
        }
        mPager.setCurrentItem(0);
        setButton(0);
    }

    private void setButton(int position) {
        ((RadioButton) mGroup.getChildAt(position)).setChecked(true);
    }

    private void init() {
        mInflater = ((Activity) mContext).getLayoutInflater();
        View view = mInflater.inflate(R.layout.banner_pager, null);
        mPager = (ViewPager) view.findViewById(R.id.vp_banner);
        mGroup = (RadioGroup) view.findViewById(R.id.rg_indicator);
        addView(view);
        dip_15 = Utils.dip2px(mContext, 15);
    }

    private Handler mHandler = new Handler();
    private Runnable mScroll = new Runnable() {
        @Override
        public void run() {
            scrollToNext();
            mHandler.postDelayed(this, mInterval);
        }
    };

    public void scrollToNext() {
        int index = mPager.getCurrentItem() + 1;
        if (mViewList.size() <= index) {
            index = 0;
        }
        mPager.setCurrentItem(index);
    }
```

```
    private class ImageAdapater extends PagerAdapter {
        @Override
        public int getCount() {
            return mViewList.size();
        }

        @Override
        public boolean isViewFromObject(View arg0, Object arg1) {
            return arg0 == arg1;
        }

        @Override
        public void destroyItem(ViewGroup container, int position, Object object) {
            container.removeView(mViewList.get(position));
        }

        @Override
        public Object instantiateItem(ViewGroup container, int position) {
            container.addView(mViewList.get(position));
            return mViewList.get(position);
        }
    }

    @Override
    public void onClick(View v) {
        int position = mPager.getCurrentItem();
        mListener.onBannerClick(position);
    }

    public void setOnBannerListener(BannerClickListener listener) {
        mListener = listener;
    }

    private BannerClickListener mListener;
    public interface BannerClickListener {
        public void onBannerClick(int position);
    }
}
```

在 Activity 代码中使用这个自定义的 Banner 控件不难，主要是先调用 setImage 方法设置图片列表，再调用 start 方法启动轮播动画，具体代码如下：

```
        mBanner = (BannerPager) findViewById(R.id.banner_pager);
        LayoutParams params = (LayoutParams) mBanner.getLayoutParams();
```



```
params.height = (int) (DisplayUtil.getSreenWidth(this) * 250f/ 640f);
mBanner.setLayoutParams(params);
ArrayList<Integer> bannerArray = new ArrayList<Integer>();
bannerArray.add(Integer.valueOf(R.drawable.banner_1));
bannerArray.add(Integer.valueOf(R.drawable.banner_2));
bannerArray.add(Integer.valueOf(R.drawable.banner_3));
bannerArray.add(Integer.valueOf(R.drawable.banner_4));
bannerArray.add(Integer.valueOf(R.drawable.banner_5));
mBanner.setImage(bannerArray);
mBanner.setOnBannerListener(this);
mBanner.start();
```

然后观察 Banner 轮播的动画效果，此时轮播到第 4 张图片，如图 7-25 所示。轮播到第 5 张图片的效果如图 7-26 所示。



图 7-25 轮播到第 4 张图片



图 7-26 轮播到第 5 张图片

7.4 增强型列表

本节介绍通过循环视图 RecyclerView 实现各种增强型列表，包括线性列表布局、普通网格布局、瀑布流网格布局等，并对循环视图进行动态更新操作。

7.4.1 循环视图 RecyclerView

如果说 TabLayout 是导航栏一节的压轴兵器，那么循环视图 RecyclerView 就是本章的终极兵器，因为功能实在是太强大了，强大到秒杀列表视图 ListView，再秒杀网格视图 GridView，还能秒杀瀑布流网格开源框架 StaggeredGridView 和 PinterestLikeAdapterView，总之学会了 RecyclerView，你的 App 武功必然提高一个层次。

因为 RecyclerView 是 5.0 之后的新增控件，所以为了兼容以前的 Adnroid 版本，在使用该控件前要修改 build.gradle，在 dependencies 节点中加入以下代码表示导入 recyclerview 库：

```
compile 'com.android.support:recyclerview-v7:25.1.0'
```

下面看看强悍的循环视图提供的常用方法。

- setAdapter：设置列表项的适配器。适配器采用 RecyclerView.Adapter。
- setLayoutManager：设置列表项的布局管理器，包括线性布局管理器 LinearLayoutManager、网格布局管理器 GridLayoutManager、瀑布流网格布局管理器 StaggeredGridLayoutManager。
- addItemDecoration：添加列表项的分割线。
- removeItemDecoration：移除列表项的分割线。
- setItemAnimator：设置列表项的增删动画。默认动画可使用系统自带的 DefaultItemAnimator。
- addOnItemTouchListener：添加列表项的触摸监听器。因为 RecyclerView 没有实现列表项的点击接口，所以开发者可通过这里的触摸监听器监控用户手势。
- removeOnItemTouchListener：移除列表项的触摸监听器。

RecyclerView 有专门的适配器类——RecyclerView.Adapter。在调用 RecyclerView 的 setAdapter 方法前，得先实现一个从 RecyclerView.Adapter 派生而来的数据适配器，用来定义列表项的布局与具体操作。下面是与 RecyclerView.Adapter 相关的常用方法。

1. 自定义适配器必须要重写的方法。

- getItemCount：获得列表项的数目。
- onCreateViewHolder：创建整个布局的视图持有者。输入参数中包括视图类型，可根据视图类型加载不同的布局，从而实现带头部的列表布局。
- onBindViewHolder：绑定每项的视图持有者。

2. 可以重写也可以不重写的方法。

- getItemViewType：返回每项的视图类型。这里返回的视图类型供 onCreateViewHolder 方法使用。
- getItemId：获得每项的编号。

3. 可以直接调用的方法。

- scrollToPosition：滚动到指定位置。
- notifyItemInserted：通知适配器在指定位置已插入新项。
- notifyItemRemoved：通知适配器在指定位置已删除原有项。
- notifyItemChanged：通知适配器在指定位置的项目已发生变化。
- notifyDataSetChanged：通知适配器整个列表的数据已发生变化。

下面是 RecyclerView.Adapter 一个派生类的代码：

```
public class LinearAdapter extends RecyclerView.Adapter<ViewHolder> implements
        OnItemClickListener, OnItemLongClickListener {
    private final static String TAG = "LinearAdapter";
    private Context mContext;
    private LayoutInflater mInflater;
    private ArrayList<GoodsInfo> mPublicArray;
```



```
public LinearAdapter(Context context, ArrayList<GoodsInfo> publicArray) {
    mContext = context;
    mInflater = LayoutInflater.from(context);
    mPublicArray = publicArray;
}

@Override
public int getItemCount() {
    return mPublicArray.size();
}

@Override
public ViewHolder onCreateViewHolder(ViewGroup vg, int viewType) {
    View v = null;
    ViewHolder holder = null;
    v = mInflater.inflate(R.layout.item_linear, vg, false);
    holder = new ItemHolder(v);
    return holder;
}

@Override
public void onBindViewHolder(ViewHolder vh, final int position) {
    ItemHolder holder = (ItemHolder) vh;
    holder.iv_pic.setImageResource(mPublicArray.get(position).pic_id);
    holder.tv_title.setText(mPublicArray.get(position).title);
    holder.tv_desc.setText(mPublicArray.get(position).desc);
    // 列表项的点击事件需要自己实现
    holder.ll_item.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (mOnItemClickListener != null) {
                mOnItemClickListener.onItemClick(v, position);
            }
        }
    });
    holder.ll_item.setOnLongClickListener(new OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
            if (mOnItemLongClickListener != null) {
                mOnItemLongClickListener.onItemLongClick(v, position);
            }
            return true;
```

```
            }
        });
    }

    @Override
    public int getItemViewType(int position) {
        // 这里返回每项的类型，开发者可自定义头部类型与一般类型
        // 在 onCreateViewHolder 方法中根据类型加载的不同布局，从而实现带头部的网格布局
        return 0;
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    public class ItemHolder extends RecyclerView.ViewHolder {
        public LinearLayout ll_item;
        public ImageView iv_pic;
        public TextView tv_title;
        public TextView tv_desc;

        public ItemHolder(View v) {
            super(v);
            ll_item = (LinearLayout) v.findViewById(R.id.ll_item);
            iv_pic = (ImageView) v.findViewById(R.id.iv_pic);
            tv_title = (TextView) v.findViewById(R.id.tv_title);
            tv_desc = (TextView) v.findViewById(R.id.tv_desc);
        }
    }
    private OnItemClickListener mOnItemClickListener;
    public void setOnItemClickListener(OnItemClickListener listener) {
        this.mOnItemClickListener = listener;
    }
    private OnItemLongClickListener mOnItemLongClickListener;
    public void setOnItemLongClickListener(OnItemLongClickListener listener) {
        this.mOnItemLongClickListener = listener;
    }
    @Override
    public void onItemClick(View view, int position) {
        String desc = String.format("您点击了第%d 项，标题是%s", position + 1,
                mPublicArray.get(position).title);
```

```
            Toast.makeText(mContext, desc, Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onItemLongClick(View view, int position) {
            String desc = String.format("您长按了第%d 项，标题是%s", position + 1,
                    mPublicArray.get(position).title);
            Toast.makeText(mContext, desc, Toast.LENGTH_SHORT).show();
        }
    }
```

下面是在活动页面中操作循环视图及其适配器的代码片段：

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_recycler_linear);
        rv_linear = (RecyclerView) findViewById(R.id.rv_linear);
        LinearLayoutManager manager = new LinearLayoutManager(this);
        manager.setOrientation(LinearLayout.VERTICAL);
        rv_linear.setLayoutManager(manager);
        LinearAdapter adapter = new LinearAdapter(this, GoodsInfo.getDefaultList());
        adapter.setOnItemClickListener(adapter);
        adapter.setOnItemLongClickListener(adapter);
        rv_linear.setAdapter(adapter);
        rv_linear.setItemAnimator(new DefaultItemAnimator());
        rv_linear.addItemDecoration(new SpacesItemDecoration(1));
    }
```

上面的代码实现的循环视图效果如图 7-27 所示。这里仿照微信公众号的消息列表，看起来像是用 ListView 实现的，当然 RecyclerView 的实际功能并不仅限于此。



图 7-27　循环视图的简单实现

7.4.2 布局管理器 LayoutManager

布局管理器 LayoutManager 是 RecyclerView 的精髓，也是 RecyclerView 强悍的源泉。LayoutManager 不但提供了 3 类布局管理，分别实现类似列表视图、网格视图、瀑布流网格的效果，而且可在代码中随时由循环视图对象调用 setLayoutManager 方法设置新的布局。一旦调用了 setLayoutManager 方法，界面就会根据新布局刷新列表项。这个特性特别适用于手机在竖屏与横屏之间的显示切换（如竖屏时展示列表，横屏时展示网格），也适用于在不同屏幕分辨率（如手机与平板）之间的显示切换（如在手机上展示列表，在平板上展示网格）。下面对这 3 类布局管理器分别进行介绍。

1. 线性布局管理器 LinearLayoutManager

LinearLayoutManager 类似于线性布局 LinearLayout，在垂直方向布局时，展示效果类似于垂直的列表视图 ListView；在水平方向布局时，展示效果类似于水平的列表视图。

下面是 LinearLayoutManager 的常用方法。

- 构造函数：可指定列表的方向和是否为相反方向开始布局。
- setOrientation：设置列表的方向，可取值 LinearLayout.HORIZONTAL 或 LinearLayout.VERTICAL。
- setReverseLayout：设置是否为相反方向开始布局，默认 false。如果设置为 true，那么垂直方向将从下往上开始布局，水平方向将从右往左开始布局。

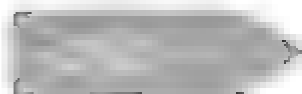
前面在介绍循环视图时采用的代码基于线性布局管理器，具体的效果如图 7-27 所示。对于令人头疼的列表项分隔线，RecyclerView 采取的做法是让开发者自定义分隔线的样式。下面是一个最简单的分隔线的实现，允许设置分隔线的宽度，代码如下：

```
public class SpacesItemDecoration extends RecyclerView.ItemDecoration {
    private int space;
    public SpacesItemDecoration(int space) {
        this.space = space;
    }

    @Override
    public void getItemOffsets(Rect outRect, View view, RecyclerView parent, RecyclerView.State state) {
        outRect.left = space;
        outRect.right = space;
        outRect.bottom = space;
        outRect.top = space;
    }
}
```

2. 网格布局管理器 GridLayoutManager

GridLayoutManager 类似于网格布局 GridLayout（该控件是 Android4.0 之后新加的）。从



展示效果来看，GridLayoutManager 类似于网格视图 GridView。所以，我们不用关心 GridLayout，把 GridLayoutManager 当成 GridView 一样使用就好了。

下面是 GridLayoutManager 的常用方法。

- 构造函数：可指定网格的列数。
- setSpanCount：设置网格的列数。
- setSpanSizeLookup：设置列表项的占位规则。默认一项占一列，如果想某项占多列，就可以在此设置自定义的占位规则，即由 GridLayoutManager.SpanSizeLookup 派生具体的实现类。

下面是在活动页面中操作网格布局管理器的示例代码：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recycler_grid);
    rv_grid = (RecyclerView) findViewById(R.id.rv_grid);
    GridLayoutManager manager = new GridLayoutManager(this, 5);
    rv_grid.setLayoutManager(manager);
    GridAdapter adapter = new GridAdapter(this, GoodsInfo.getDefaultGrid());
    adapter.setOnItemClickListener(adapter);
    adapter.setOnItemLongClickListener(adapter);
    rv_grid.setAdapter(adapter);
    rv_grid.setItemAnimator(new DefaultItemAnimator());
    rv_grid.addItemDecoration(new SpacesItemDecoration(1));
}
```

网格布局管理器的效果如图 7-28 所示，看起来跟 GridView 的展示效果没什么区别。



图 7-28 循环视图的网格布局

但绝非 GridView 可比，因为网格布局管理器提供了 setSpanSizeLookup 方法，该方法允许一个网格占据多列空间，更加灵活易用。下面是使用占位规则的 Activity 代码片段：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recycler_combine);
    rv_combine = (RecyclerView) findViewById(R.id.rv_combine);
    GridLayoutManager manager = new GridLayoutManager(this, 4);
```

```
        //以下占位规则的意思是：第一项和第二项占两列，其他项占一列
        //如果网格的列数为4，那么第一项和第二项平分第一行，从第二行开始每行有4项
        manager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {
            @Override
            public int getSpanSize(int position) {
                if (position == 0 || position==1) {
                    return 2;
                } else {
                    return 1;
                }
            }
        });
        rv_combine.setLayoutManager(manager);
        CombineAdapter adapter = new CombineAdapter(this, GoodsInfo.getDefaultCombine());
        adapter.setOnItemClickListener(adapter);
        adapter.setOnItemLongClickListener(adapter);
        rv_combine.setAdapter(adapter);
        rv_combine.setItemAnimator(new DefaultItemAnimator());
        rv_combine.addItemDecoration(new SpacesItemDecoration(1));
    }
```

使用占位规则的效果如图 7-29 所示。可以看到，第一行只有两个网格，第二行有 4 个网格，这意味着第一行的每个网格都占据了两列位置。



图 7-29　循环视图的合并网格布局效果

3. 瀑布流网格布局管理器 StaggeredGridLayoutManager

电商 App 在展示众多商品信息时，往往使用灵活高度的格子展示。因为不同商品的外观尺寸不一样，比如冰箱高高的纵向比较长，空调横向比较长，所以若用一样规格的网格展示，必然有的商品图片会被压缩得很小。这种情况得根据不同的商品形状展示不同高度的图片，这就是瀑布流网格的应用场合。StaggeredGridLayoutManager 让瀑布流效果的开发大大简化了，只要在适配器中动态设置每个网格的高度，系统就会自动在界面上依次排列瀑布流网格。

下面是 StaggeredGridLayoutManager 的常用方法。

- 构造函数：可指定网格的列数和方向。
- setSpanCount :设置网格的列数。
- setOrientation :设置瀑布流布局的方向。取值说明同 LinearLayoutManager。
- setReverseLayout：设置是否为相反方向开始布局，默认 false。如果设置为 true，那么垂直方向将从下往上开始布局，水平方向将从右往左开始布局。

下面是在活动页面中操作瀑布流网格布局管理器的示例代码：

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_recycler_staggered);
        rv_staggered = (RecyclerView) findViewById(R.id.rv_staggered);
        StaggeredGridLayoutManager manager = new StaggeredGridLayoutManager(3, LinearLayout.
VERTICAL);
        rv_staggered.setLayoutManager(manager);
        StaggeredAdapter adapter = new StaggeredAdapter(this, GoodsInfo.getDefaultStag());
        adapter.setOnItemClickListener(adapter);
        adapter.setOnItemLongClickListener(adapter);
        rv_staggered.setAdapter(adapter);
        rv_staggered.setItemAnimator(new DefaultItemAnimator());
        rv_staggered.addItemDecoration(new SpacesItemDecoration(3));
    }
```

瀑布流网格布局的效果如图 7-30 与图 7-31 所示，每个网格的高度依照具体图片的高度变化而变化，整个页面看起来变得生动活泼。读者可以打开淘宝 App，在顶部导航栏搜索“连衣裙”，看看搜索结果页面是不是如瀑布流网格这般交错显示？



图 7-30　循环视图的瀑布流效果 1



图 7-31　循环视图的瀑布流效果 2

7.4.3 动态更新循环视图

循环视图之所以成为终极兵器，不单单因为具备列表视图、网格视图、瀑布流网格三者的功力，更是因为允许动态更新内部数据。不但可以单独更新某项视图，而且能够顺便展示增删动画，好比刀光剑影起落之际还在演奏乐曲，这才是真正的无招胜有招。

下面是在 Acitivity 页面中对循环视图内部数据进行动态增、删、改的代码片段：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_recycler_dynamic);
    findViewById(R.id.btn_recycler_add).setOnClickListener(this);
    rv_dynamic = (RecyclerView) findViewById(R.id.rv_dynamic);
    LinearLayoutManager manager = new LinearLayoutManager(this);
    manager.setOrientation(LinearLayout.VERTICAL);
    rv_dynamic.setLayoutManager(manager);
    mAllArray = GoodsInfo.getDefaultList();
    mPublicArray = GoodsInfo.getDefaultList();
    mAdapter = new LinearDynamicAdapter(this, mPublicArray);
    mAdapter.setOnItemClickListener(this);
    mAdapter.setOnItemLongClickListener(this);
    mAdapter.setOnItemDeleteClickListener(this);
    rv_dynamic.setAdapter(mAdapter);
    rv_dynamic.setItemAnimator(new DefaultItemAnimator());
    rv_dynamic.addItemDecoration(new SpacesItemDecoration(1));
}

@Override
public void onClick(View v) {
    if (v.getId() == R.id.btn_recycler_add) {
        int position = (int) (Math.random()*100%mAllArray.size());
        GoodsInfo old_item = mAllArray.get(position);
        GoodsInfo new_item = new GoodsInfo(old_item.pic_id, old_item.title, old_item.desc);
        mPublicArray.add(0, new_item);
        mAdapter.notifyItemInserted(0);
        rv_dynamic.scrollToPosition(0);
    }
}

@Override
public void onItemLongClick(View view, int position) {
    GoodsInfo item = mPublicArray.get(position);
    item.bPressed = !item.bPressed;
    mPublicArray.set(position, item);
```

```
            mAdapter.notifyItemChanged(position);
        }

        @Override
        public void onItemClick(View view, int position) {
            String desc = String.format("您点击了第%d 项，标题是%s", position + 1,
                    mPublicArray.get(position).title);
            Toast.makeText(this, desc, Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onItemDeleteClick(View view, int position) {
            mPublicArray.remove(position);
            mAdapter.notifyItemRemoved(position);
        }
```

具体的演示效果如图 7-32、图 7-33、图 7-34、图 7-35 所示。其中，图 7-32 所示为页面的初始截图；在列表顶部新增一条消息的截图，消息添加时其实是有动画的，图 7-33 所示为动画结束之后的界面；图 7-34 所示为长按某条消息时的截图，有 iphone 的同学可以打开微信，长按里面的某条聊天记录，看看是不是在记录右边弹出“删除该聊天”按钮；点击“删除该聊天”会展示记录的删除动画，动画结束的界面如图 7-35 所示。

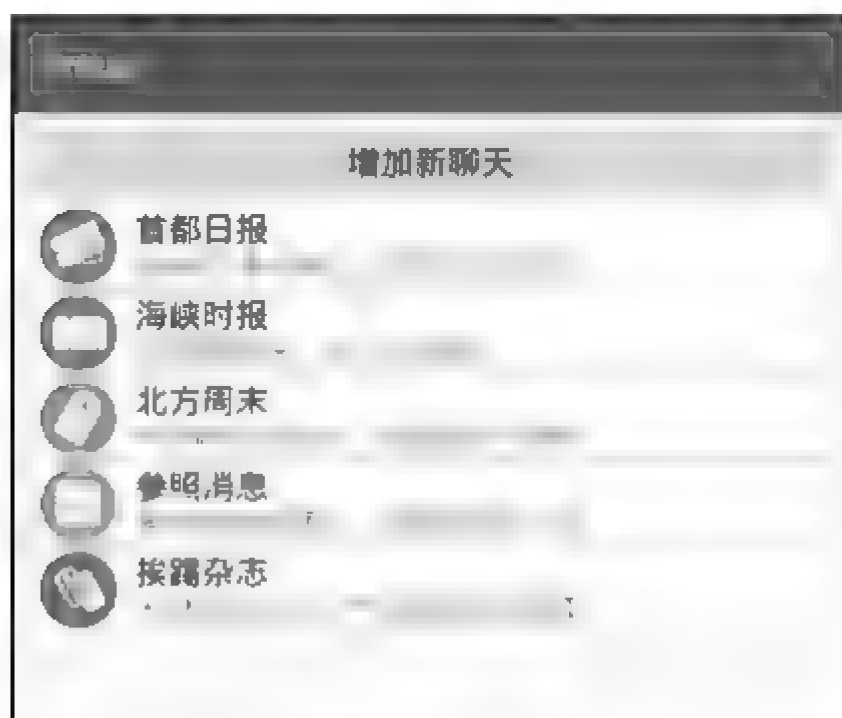


图 7-32　消息的初始页面



图 7-33　新增了一条消息



图 7-34　长按某条消息的页面



图 7-35　删除该消息的页面

7.5 实战项目：仿淘宝主页

各位亲爱的读者，经过艰苦的 App 开发学习，终于来到了本节的实战项目"仿淘宝主页"。淘宝 App 的主页动感十足，页面元素丰富，令人眼花缭乱，其中运用了 Android 的多种终极兵器，可谓是 App 开发 UI 的集大成之作。其实到目前为止，本章的知识点已经涵盖了淘宝主页的大部分技术，所以仿照淘宝主页做一个山寨的电商 App 首页也不是什么难事，接下来让我们好好分析一下如何实现。



图 7-36 淘宝主页截图

7.5.1 设计思路

首先看看大家都熟悉的淘宝主页长什么模样，如图 7-36 所示。是不是很熟悉呢？其实该页面是各电商 App 首页的通用模板。除了淘宝外，还有京东、苏宁易购、当当、美团、百度糯米等，这些电商 App 的主页都大同小异，所以只要吃透了淘宝主页采用的 App 技术，其他电商 App 也能依葫芦画瓢。

因为我们的实战项目只是仿淘宝主页，而不是完全一模一样，所以页面只要大致相似就行。下面是两张山寨后的页面效果，图 7-37 所示为首页页面的效果图，图 7-38 所示为分类页面的效果图。



图 7-37 仿淘宝的首页页面



图 7-38 仿淘宝的分类页面

数数这两张效果图分别运用了本章的哪些知识点。这两个页面基本上是由前面介绍的各控件效果图拼接而成的，找起来也不难。

- 标签栏 Tabbar：页面底部有一排标签按钮。
- 工具栏 Toolbar：页面顶部的导航栏是工具栏 Toolbar。
- 溢出菜单 OverflowMenu：页面右上角的三点按钮是标准的溢出菜单提示。
- 搜索框 SearchView：三点按钮左边的放大镜按钮是熟悉的搜索图标。
- 横幅轮播 Banner：导航栏下方的广告图片底部有指示器，毫无疑问是 Banner。
- 循环视图 RecyclerView 的网格布局：Banner 下方的两排图标是标准的网格布局，再下面的推荐栏目是合并网格后的网格布局。
- 标签布局 TabLayout：分类页面顶部的“服装”和“电器”标签用到了标签布局。
- 循环视图 RecyclerView 的瀑布流布局：电器商品的交错展示运用了瀑布流网格布局。

另外，这个仿淘宝主页使用了前几章学过的控件，包括翻页视图 ViewPager、碎片 Fragment 等，正好一起复习。同时，购物车页面的具体处理已经体现在第 4 章的实战项目中了，有兴趣的读者可以将其整合进来，形成一个电商 App 的完整 demo。

7.5.2 小知识：下拉刷新 SwipeRefreshLayout

电商 App 在商品列表页面往往提供下拉刷新功能，把页面整体下拉即可触发页面刷新操作。Android 提供了下拉刷新控件 SwipeRefreshLayout，可用于简单的下拉刷新。

下面是 SwipeRefreshLayout 的常用方法说明。

- setOnRefreshListener：设置刷新监听器。需要重写监听器 OnRefreshListener 的 onRefresh 方法，该方法在下拉松开时触发。
- setRefreshing：设置刷新的状态。true 表示正在刷新，false 表示结束刷新。
- isRefreshing：判断是否正在刷新。
- setColorSchemeColors：设置进度圆圈的圆环颜色。
- setProgressBackgroundColorSchemeColor：设置进度圆圈的背景颜色。
- setProgressViewOffset：设置进度圆圈的偏移量。第一个参数表示进度圈是否缩放，第二个参数表示进度圈开始出现时距顶端的偏移，第三个参数表示进度圈拉到最大时距顶端的偏移。
- setDistanceToTriggerSync：设置手势向下滑动多少距离才会触发刷新操作。

需要注意的是，SwipeRefreshLayout 节点下面只能有一个直接子视图。如果有多个直接子视图，那么只会展示第一个子视图，后面的子视图将不予展示。这个直接子视图必须允许滚动，比如 ScrollView、ListView、GridView、RecyclerView 等。如果不是这些视图，就不支持滚动，更不支持下拉刷新。下面是在布局文件中使用 SwipeRefreshLayout 的代码：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="5dp" >

    <android.support.v4.widget.SwipeRefreshLayout
```



```
        android:id="@+id/srl_simple"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <ScrollView
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >

            <TextView
                android:id="@+id/tv_simple"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:paddingTop="10dp"
                android:text="这是一个简单视图"
                android:textColor="#000000"
                android:textSize="17sp" />
        </ScrollView>
    </android.support.v4.widget.SwipeRefreshLayout>
</LinearLayout>
```

与上面的布局文件对应的完整 Activity 代码如下：

```
public class SwipeRefreshActivity extends AppCompatActivity implements OnRefreshListener {
    private TextView tv_simple;
    private SwipeRefreshLayout srl_simple;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_swipe_refresh);
        tv_simple = (TextView) findViewById(R.id.tv_simple);
        srl_simple = (SwipeRefreshLayout) findViewById(R.id.srl_simple);
        srl_simple.setOnRefreshListener(this);
        srl_simple.setColorSchemeResources(R.color.red, R.color.orange, R.color.green, R.color.blue);
    }

    @Override
    public void onRefresh() {
        tv_simple.setText("正在刷新");
        mHandler.postDelayed(mRefresh, 2000);
    }

    private Handler mHandler = new Handler();
```

```
        private Runnable mRefresh = new Runnable() {
            @Override
            public void run() {
                tv_simple.setText("刷新完成");
                srl_simple.setRefreshing(false);
            }
        };
    }
```

这个简单下拉刷新的效果如图 7-39 和图 7-40 所示。其中，图 7-39 所示为开始刷新时的截图，图 7-40 所示为结束刷新时的截图。



图 7-39　开始刷新时的截图



图 7-40　结束刷新时的截图

SwipeRefreshLayout 更好的用法是与 RecyclerView 相结合，通过下拉刷新操作动态添加循环视图的记录，从而省去一个添加按钮或刷新按钮，就优化用户体验来说，避免按钮太多而显得凌乱。下面是在活动页面中结合 SwipeRefreshLayout 与 RecyclerView 的代码片段：

```
        @Override
        public void onRefresh() {
            mHandler.postDelayed(mRefresh, 2000);
        }

        private Handler mHandler = new Handler();
        private Runnable mRefresh = new Runnable() {
            @Override
            public void run() {
                srl_dynamic.setRefreshing(false);
                int position = (int) (Math.random() * 100 % mAllArray.size());
                GoodsInfo old_item = mAllArray.get(position);
                GoodsInfo new_item = new GoodsInfo(old_item.pic_id, old_item.title, old_item.desc);
                mPublicArray.add(0, new_item);
                mAdapter.notifyItemInserted(0);
                // 当循环视图的列表项已经占满整个屏幕时，再往顶部添加一条新记录，感觉屏幕没
有发生变化，也没看到插入动画。此时要调用 scrollToPosition(0)方法，表示滚动到第一条记录。
                rv_dynamic.scrollToPosition(0);
            }
        };
```

对循环视图进行下拉刷新的效果如图 7-41 和图 7-42 所示。其中，图 7-41 所示为开始刷

新时的列表界面；图 7-42 所示为结束刷新时的列表界面，此时列表顶端增加了一条新记录。



图 7-41　刷新中的消息列表



图 7-42　刷新完成的消息列表

7.5.3　代码示例

本章的实战项目用到了 TabLayout 与 RecyclerView，因为这两个控件都需要导入对应的库，所以编码过程与前两章相比多了一步，共分为 6 步。

01 设计代码架构，初步拆分后的 package 包，包括以下 6 部分。

- com.example.department.activity：存放 Acitivity 页面的代码。
- com.example.department.adapter：存放适配器的代码。
- com.example.department.bean：存放实体数据结构的代码，如商品信息。
- com.example.department.fragment：存放碎片代码。
- com.example.department.util：存放工具类代码。
- com.example.department.widget：存放自定义控件的代码。

02 想好代码文件与布局文件的名称，比如 App 主页面的代码文件取名 DepartmentStoreActivity.java，对应的布局文件名是 activity_department_store.xml，其下有 3 个子页面，包括首页页面的代码文件取名 DepartmentHomeActivity.java，对应的布局文件名是 activity_department_home.xml；分类页面的代码文件取名 DepartmentClassActivity.java，对应的布局文件名是 activity_department_class.xml；购物车页面的代码文件取名 DepartmentCartActivity.java，对应的布局文件名是 activity_department_cart.xml。

除此之外，还有搜索页面 SearchViewActivity、搜索结果页面 SearchResultActvity 以及碎片、适配器的代码及其布局文件，读者可自行构思。

03 打开 build.gradle，在 dependencies 节点中加入下面 3 行代码，表示分别导入 appcompat-v7、design、recyclerview-v7 三个库：

```
compile 'com.android.support:appcompat-v7:25.1.0'
compile 'com.android.support:design:25.1.0'
compile 'com.android.support:recyclerview-v7:25.1.0'
```

步骤 04 在 AndroidManifest.xml 中补充相应配置，主要有以下两点：

（1）注册两个页面的 acitivity 节点，注册代码如下：

```
<activity android:name=".DepartmentStoreActivity" android:theme="@style/AppCompatTheme" />
<activity android:name=".DepartmentHomeActivity" android:theme="@style/AppCompatTheme" />
<activity android:name=".DepartmentClassActivity" android:theme="@style/AppCompatTheme" />
<activity android:name=".DepartmentCartActivity" android:theme="@style/AppCompatTheme" />
<activity android:name=".SearchViewActivity" android:theme="@style/AppCompatTheme" />
```

（2）对 SearchResultActvity 单独配置，注册代码举例如下：

```
<activity android:name=".SearchResultActvity" android:theme="@style/AppCompatTheme" >
    <intent-filter>
        <action android:name="android.intent.action.SEARCH"/>
    </intent-filter>
    <meta-data android:name="android.app.searchable" android:resource="@xml/searchable"/>
</activity>
```

注意这里给 activity 节点补充了 AppCompatTheme 风格，目的是声明不带 ActionBar 的风格，以便 Activity 代码内部使用 Toolbar 替换 ActionBar。

步骤 05 在资源目录下补充相应的 XML 配置，包括以下 5 点：

（1）在 res/drawable 目录下存放相关状态图形的描述文件。
（2）在 res/layout 目录下编写页面、碎片、适配器、标签页等对应的布局文件。
（3）在 res/menu 目录下编写溢出菜单的布局文件。
（4）在 res/values/styles.xml 中补充 AppCompatTheme 与标签按钮的样式定义。
（5）在 res/xml 目录下创建 searchable.xml，编写根节点为 searchable 的搜索框样式定义。

步骤 06 进行 java 代码开发，包括对页面、碎片、适配器等进行编码。

下面是首页页面 DepartmentHomeActivity.java 的完整代码：

```
public class DepartmentHomeActivity extends AppCompatActivity implements BannerClickListener {
    private final static String TAG = "DepartmentHomeActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_department_home);
        Toolbar tl_head = (Toolbar) findViewById(R.id.tl_head);
        tl_head.setTitle("商城首页");
        setSupportActionBar(tl_head);
        initBanner();
        initGrid();
        initCombine();
    }
```



```
    private void initBanner() {
        BannerPager banner = (BannerPager) findViewById(R.id.banner_pager);
        LayoutParams params = (LayoutParams) banner.getLayoutParams();
        params.height = (int) (DisplayUtil.getSreenWidth(this) * 250f/ 640f);
        banner.setLayoutParams(params);
        ArrayList<Integer> bannerArray = new ArrayList<Integer>();
        bannerArray.add(Integer.valueOf(R.drawable.banner_1));
        bannerArray.add(Integer.valueOf(R.drawable.banner_2));
        bannerArray.add(Integer.valueOf(R.drawable.banner_3));
        bannerArray.add(Integer.valueOf(R.drawable.banner_4));
        bannerArray.add(Integer.valueOf(R.drawable.banner_5));
        banner.setImage(bannerArray);
        banner.setOnBannerListener(this);
        banner.start();
    }

    @Override
    public void onBannerClick(int position) {
        String desc = String.format("您点击了第%d 张图片", position+1);
        Toast.makeText(this, desc, Toast.LENGTH_LONG).show();
    }

    private void initGrid() {
        RecyclerView rv_grid = (RecyclerView) findViewById(R.id.rv_grid);
        GridLayoutManager manager = new GridLayoutManager(this, 5);
        rv_grid.setLayoutManager(manager);
        GridAdapter adapter = new GridAdapter(this, GoodsInfo.getDefaultGrid());
        adapter.setOnItemClickListener(adapter);
        adapter.setOnItemLongClickListener(adapter);
        rv_grid.setAdapter(adapter);
        rv_grid.setItemAnimator(new DefaultItemAnimator());
        rv_grid.addItemDecoration(new SpacesItemDecoration(1));
    }

    private void initCombine() {
        RecyclerView rv_combine = (RecyclerView) findViewById(R.id.rv_combine);
        GridLayoutManager manager = new GridLayoutManager(this, 4);
        manager.setSpanSizeLookup(new GridLayoutManager.SpanSizeLookup() {
            @Override
            public int getSpanSize(int position) {
                if (position == 0 || position==1) {
                    return 2;
```

```
                } else {
                    return 1;
                }
            }
        });
        rv_combine.setLayoutManager(manager);
        CombineAdapter adapter = new CombineAdapter(this, GoodsInfo.getDefaultCombine());
        adapter.setOnItemClickListener(adapter);
        adapter.setOnItemLongClickListener(adapter);
        rv_combine.setAdapter(adapter);
        rv_combine.setItemAnimator(new DefaultItemAnimator());
        rv_combine.addItemDecoration(new SpacesItemDecoration(1));
    }

    @Override
    public boolean onMenuOpened(int featureId, Menu menu) {
        Utils.setOverflowIconVisible(featureId, menu);  // 显示菜单项左侧的图标
        return super.onMenuOpened(featureId, menu);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_home, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == android.R.id.home) {
            finish();
        } else if (id == R.id.menu_search) {
            Intent intent = new Intent(this, SearchViewActivity.class);
            intent.putExtra("collapse", false);
            startActivity(intent);
        } else if (id == R.id.menu_refresh) {
            Toast.makeText(this, "当前刷新时间："+Utils.getNowDateTime("yyyy-MM-dd
HH:mm:ss"), Toast.LENGTH_LONG).show();
            return true;
        } else if (id == R.id.menu_about) {
            Toast.makeText(this, "这个是商城首页", Toast.LENGTH_LONG).show();
            return true;
        } else if (id == R.id.menu_quit) {
```

```
                finish();
            }
            return super.onOptionsItemSelected(item);
        }
    }
```

7.6 小　　结

本章主要介绍了 App 开发的组合控件相关知识，包括标签栏的用法（标签按钮、3 种标签栏的实现方式）、导航栏的用法（工具栏、溢出菜单、搜索框、标签布局）、横幅条的用法（自定义指示器、横幅轮播 Banner 的实现）、增强型列表的用法（循环视图、3 种布局管理器、动态变更循环视图）。最后设计了一个实战项目“仿淘宝主页”，在该项目的 App 编码中采用了本章介绍的大部分组合控件知识，并复习了前几章的相关技术。另外，介绍了如何使用下拉刷新控件。

通过本章的学习，读者应该能够掌握以下 4 种开发技能：

（1）学会底部标签栏的实现与用法。
（2）学会顶部导航栏的实现与用法。
（3）学会横幅轮播 Banner 的实现与用法。
（4）学会循环视图及其 3 种布局管理器的用法，以及通过下拉刷新控件动态更新视图记录。



调试与上线

本章介绍 App 从调试到上线的完整过程，主要包括利用模拟器和真机调试 App、App 在上线前的各种准备工作、对 App 安装包进行安全加固、把 App 发布到应用商店的具体步骤等。

8.1 调试工作

本节介绍几种常见的 App 调试方法，包括使用外置模拟器调试，比如几种国产模拟器的用法；电脑连接真机调试，描述真机调试要具备的条件；分发 APK 安装包给他人调试，着重说明签名证书的创建方法，以及如何利用签名证书导出 APK 安装包。

8.1.1 模拟器调试

前面几章的 App 开发学习基本采用了模拟器进行功能测试与效果演示。在模拟器的使用过程中，不知道读者有没有发现 Android Studio 自带的模拟器存在诸多不便，比如：

（1）内置模拟器启动速度慢，资源占用大。

（2）单个模拟器的屏幕分辨率是固定的，如果要测试不同分辨率，就只能另外创建新的模拟器。

（3）内置模拟器默认是竖屏显示，无法测试横屏的显示效果。

（4）内置模拟器不支持设置手机信息，如手机品牌、型号、IMEI 等。

（5）内置模拟器不支持模拟传感器功能，如摇一摇等。

（6）内置模拟器不支持模拟定位。

从上面可以看出，内置模拟器用于简单 App 的测试还凑合，如果用于高级测试场景就无法胜任。为了方便广大 App 开发者，各种外置的安卓模拟器如雨后春笋般涌现出来，比如国外的 Genymotion 模拟器、各种国产模拟器。这里笔者介绍两款用得比较多的国产模拟器——逍遥安卓模拟器和夜神模拟器。

1. 逍遥安卓模拟器

逍遥安卓模拟器基于 Android4.2.2（SDK 版本 19），官方网站地址是 http://www.xyaz.cn/。从官方网站下载安装文件，下载完成后双击即可弹出安装界面，如图 8-1 所示。



图 8-1　逍遥安卓的安装界面

选择模拟器的安装目录路径，然后单击“安装”按钮，等待安装过程。安装结束后，桌面会出现名为“逍遥安卓”的图标，双击该图标打开模拟器，模拟器的启动需要一定时间（几十秒到数分钟，与电脑性能有关）。耐心等待模拟器启动完毕，界面切换到模拟器的仿手机界面主页，如图 8-2 所示。

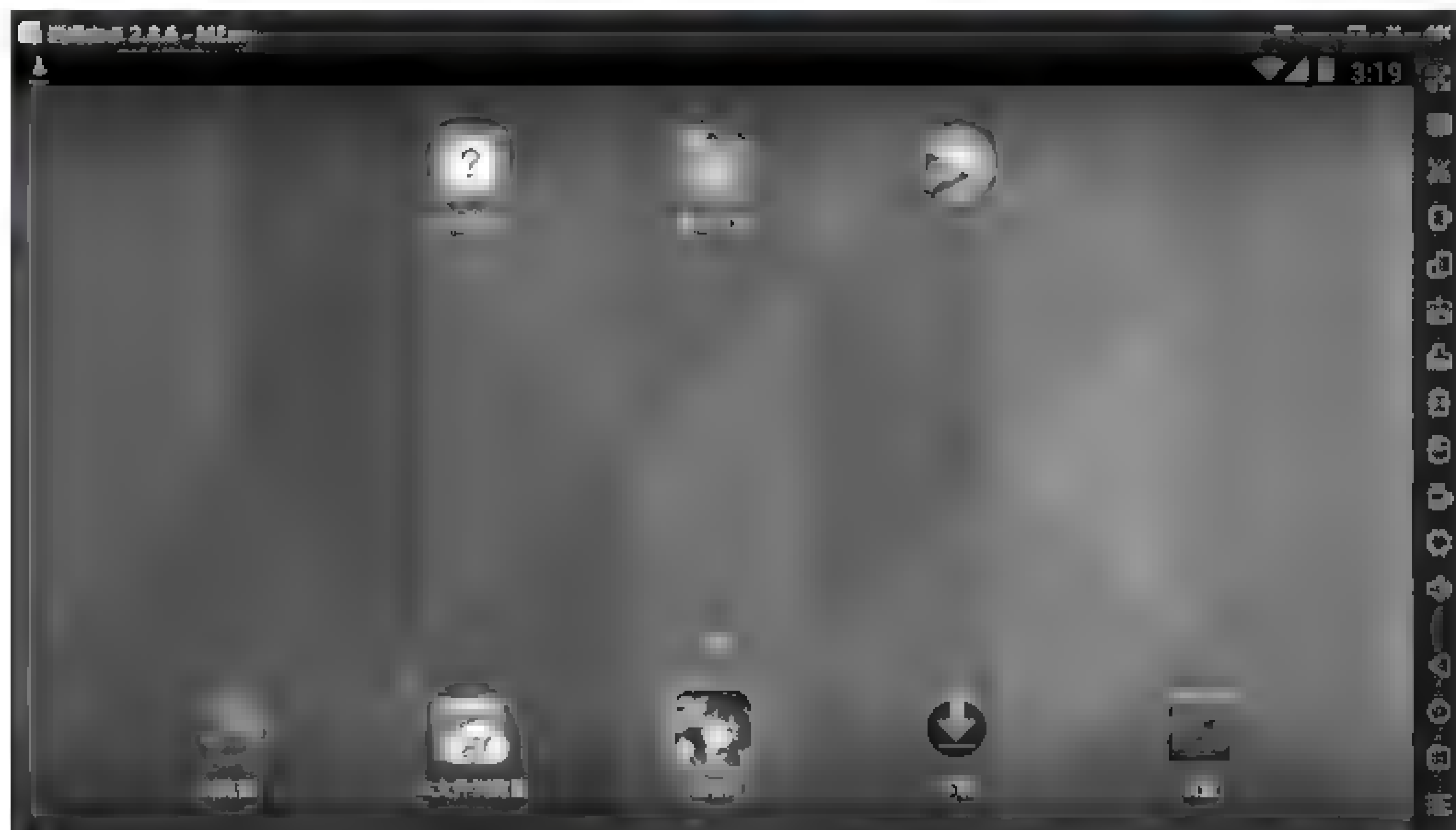


图 8-2　逍遥安卓的横屏桌面

逍遥安卓默认展示横屏，若想切换到竖屏显示，则单击模拟器主界面右侧一列图标的第 5 个或第 6 个“旋转屏幕”图标，即可进行横竖屏切换，切换后的竖屏界面如图 8-3 所示。

右侧图标列除了“旋转屏幕”外，还有截图、摇一摇、屏幕录制、设置（齿轮图标）、音量控制等图标。单击齿轮图标打开设置窗口，在“常用”页面可设置 CPU 个数、内存大小、分辨率等信息，在“高级”页面可设置手机品牌、手机型号、IMEI 串号等信息，如图 8-4 所示。设置修改完毕后，单击窗口下方的“保存”按钮，新设置在下次启动模拟器后生效。



图 8-3　逍遥安卓的竖屏桌面



图 8-4　逍遥安卓的设置界面

逍遥安卓主界面右下角还有一列（共 4 个图标），从上往下依次表示后退键、桌面键、任务键、菜单键。多数手机的下方只有三个按键，从左往右分别是菜单键、桌面键、后退键，长按桌面键会弹出任务列表（近期有运行的 App 列表），有的手机取消菜单键换成任务键，逍遥安卓提供 4 个按钮模拟这 4 种按键操作。

利用逍遥安卓调试 App 的过程与内置模拟器类似，开发者先启动 Android Studio，等待启动完成后再双击启动逍遥安卓。等待逍遥安卓启动完成进入桌面后，在 Android Studio 上依次选择菜单 Run→Run '***'，这时弹出设备选择窗口，如图 8-5 所示。

该窗口中的 Samsung GT-P52100（Android 4.2.2, API 17）为逍遥安卓模拟器，单击窗口下方的 OK 按钮，等待 Android Studio 编译并将 App 安装到逍遥安卓，后续的 App 调试操作就可以在模拟器上执行了。

2. 夜神模拟器

夜神模拟器基于 Android4.4.2（SDK 版本 17），官方网站地址是 http://www.yeshen.com/。从官方网站下载安装文件，下载完成后双击打开，弹出安装界面如图 8-6 所示。

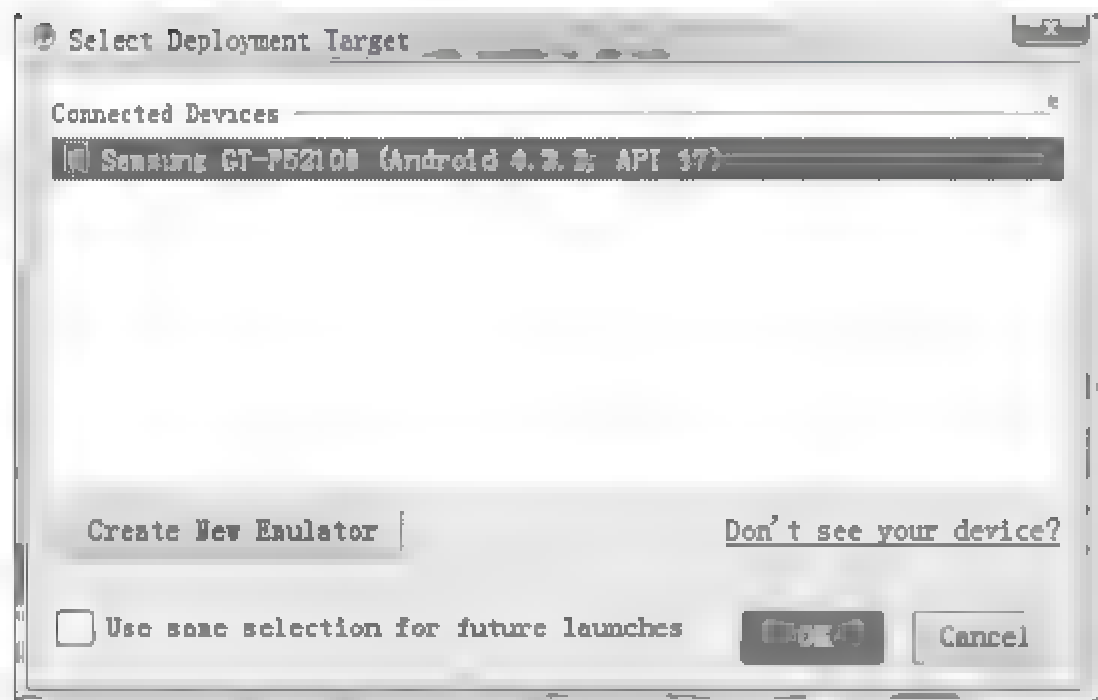


图 8-5 启动 App 时发现逍遥安卓模拟器



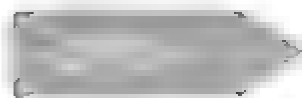
图 8-6 夜神模拟器的安装界面

单击“快速安装”按钮或右下角的“自定义安装”对安装路径进行设置。在安装过程中，系统可能会弹出对话框提示是否安装设备软件，提示对话框如图 8-7 所示。



图 8-7 安装设备软件的提示对话框

在该对话框单击“安装”按钮，等待安装过程。安装结束后，桌面会出现名为“夜神模



拟器”的图标，双击该图标打开模拟器，模拟器的启动需要一定时间。耐心等待模拟器启动完毕，界面切换到模拟器的仿手机界面主页，如图 8-8 所示。



图 8-8　夜神模拟器的横屏桌面

夜神默认展示竖屏还是横屏与设置有关，在界面右上角的中央找到一个齿轮图标，这是模拟器的设置入口，单击该图标弹出设置窗口，如图 8-9 所示。



图 8-9　夜神模拟器的设置界面

在设置窗口的左边菜单列表中单击“高级菜单”选项，窗口右边就切换到高级设置页面。这里可以设置模拟的 CPU 个数、内存大小，还可设置默认显示横屏（平板版）还是竖屏（手机版），以及模拟界面的屏幕分辨率。设置修改完成后，单击窗口下方的“保存设置”按钮，下次启动模拟器时就会生效最新的设置。

夜神模拟器主界面右边是一列图标按钮，用于一些特殊功能的快捷操作，包括摇一摇、屏幕截图、虚拟定位、音量控制、视频录制等，开发者可在具体的功能测试时加以控制。主界面右下角有 3 个控制图标，从上往下依次表示返回键、主页键、任务键（提示菜单键，其实是

任务键）。这里找不到可用的菜单键是不是很奇怪？其实夜神模拟器的菜单键需要电脑键盘输入，电脑键盘右下方的 Alt 键与 Ctrl 键之间有一个“一口三横”键（菜单键），按电脑键盘的菜单键相当于模拟器的菜单键点击操作。

利用夜神模拟器调试 App 的过程与逍遥安卓类似，开发者先启动 Android Studio，再启动夜神模拟器，等待夜神启动完毕进入桌面后，回到 Android Studio 选择菜单 Run→Run '***'。此时弹出的设备选择窗口如图 8-10 所示。

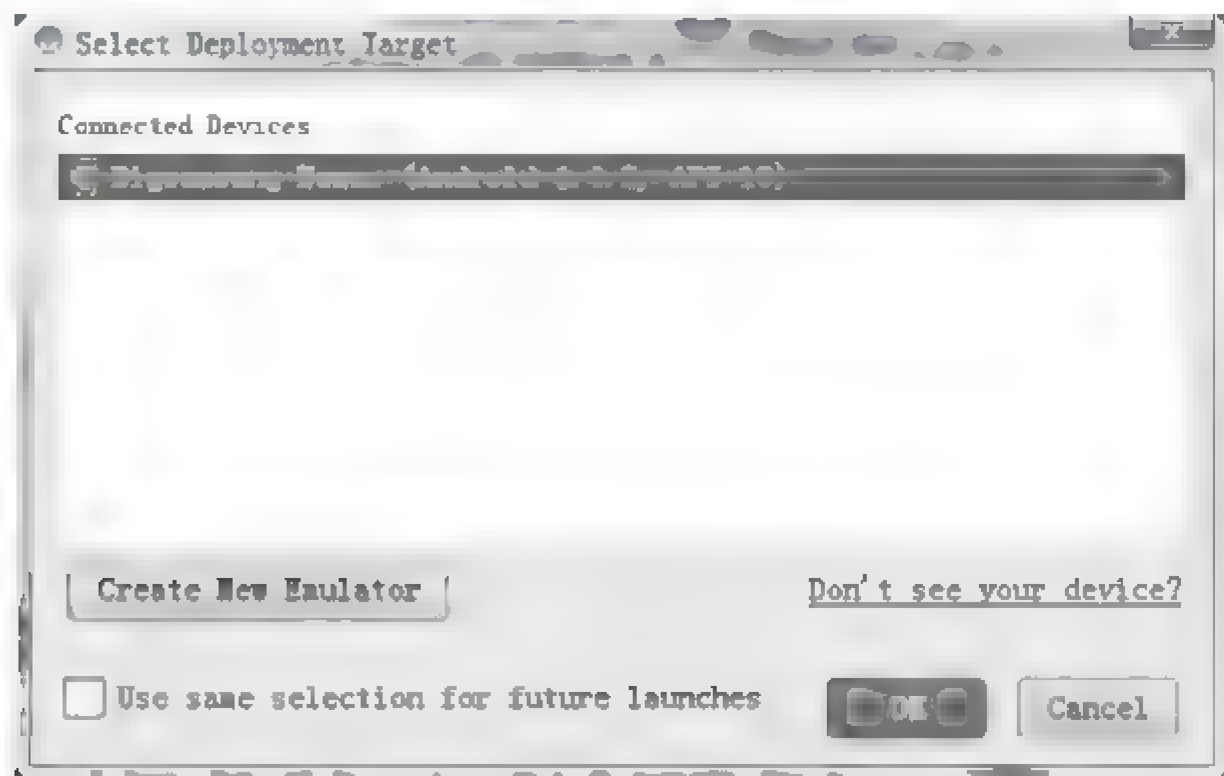


图 8-10　启动 App 时发现夜神模拟器

该窗口中的 Bigsamsung Nexus（Android 4.4.2, API 19）为夜神模拟器，单击窗口下方的 OK 按钮，等待 Android Studio 将 App 安装到夜神，后续即可在模拟器上调试。

8.1.2　真机调试

外置模拟器即使做得再好，对很多功能的调试也力有不逮，毕竟没法完全模拟真实手机，若有可能，还是尽量使用真机进行测试。利用真机调试要具备以下 4 个条件：

1. 需要使用数据线把手机连到电脑上。

手机的电源线拔掉插头就是数据线。数据线长方形的一端接到电脑的 USB 口上，即可完成手机与电脑的连接。

2. 要在电脑上安装手机的驱动程序。

一般电脑会把手机当作 USB 存储设备一样安装驱动，大多数情况会自动安装成功。如果遇到少数情况安装失败，就可以先安装 91 手机助手，自动下载并安装对应的手机驱动。

3. 要在手机上启用 USB 调试功能。

手机连接电脑后，下拉通知栏通常会有“USB 计算机连接”选项，点击该选项跳到 USB 连接页面。勾选该页面上的“USB 调试”选项开启 USB 调试功能，如图 8-11 所示。

USB 调试功能开启后，每次拿手机连接电脑，屏幕上都会弹出对话框“允许 USB 调试吗？”，如图 8-12 所示。勾选该对话框上的“一律允许使用这台计算机进行调试”，然后点击“确定”按钮，该手机就具备了 App 调试条件。



图 8-11　USB 计算机连接页面



图 8-12　USB 调试的提示对话框

USB 调试功能可在设置中通过“系统”→“开发者选项”（有的手机在“系统”→“更多设置”→“开发者选项”）找到，勾选该功能开启 USB 调试，如图 8-13 所示。

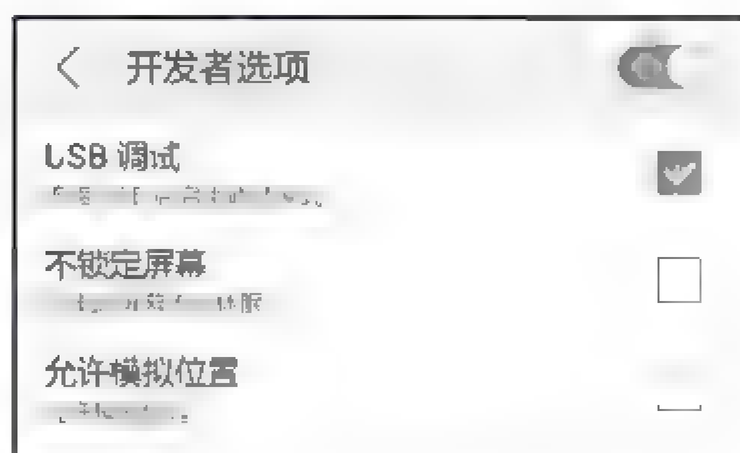


图 8-13　开发者选项页面

现在很多手机默认没有“开发者选项”这个菜单，即使把手机连接到电脑，仍然无法找到“开发者选项”，更别提 USB 调试了。此时要进入“系统”→“关于手机”→“版本信息”页面，这里有好几个版本项，每个版本项都使劲点击七、八下，总会有某个版本点击后出现“你将开启开发者模式”的提示。继续点击该版本项开启开发者模式，然后退出并重新进入设置页面，此时就能在“系统”菜单下找到“开发者选项”了。

4. 手机要处于使用状态，即不能锁屏。

锁屏状态下，Android Studio 向手机安装 App 的行为会被拦截，所以要保证手机处于解锁状态，才能顺利通过开发电脑安装 App 到手机上。

经过以上步骤，总算具备通过电脑在手机上安装 App 的条件了。马上启动 Android Studio，依次选择菜单 Run→Run '***'，在弹出的设备选择窗口可以看到已连接的手机信息，如图 8-14 所示。此时的设备信息提示这是一台小米手机，单击窗口下方的 OK 按钮，接下来的事情就是等待 Android Studio 往手机上安装 App 了。

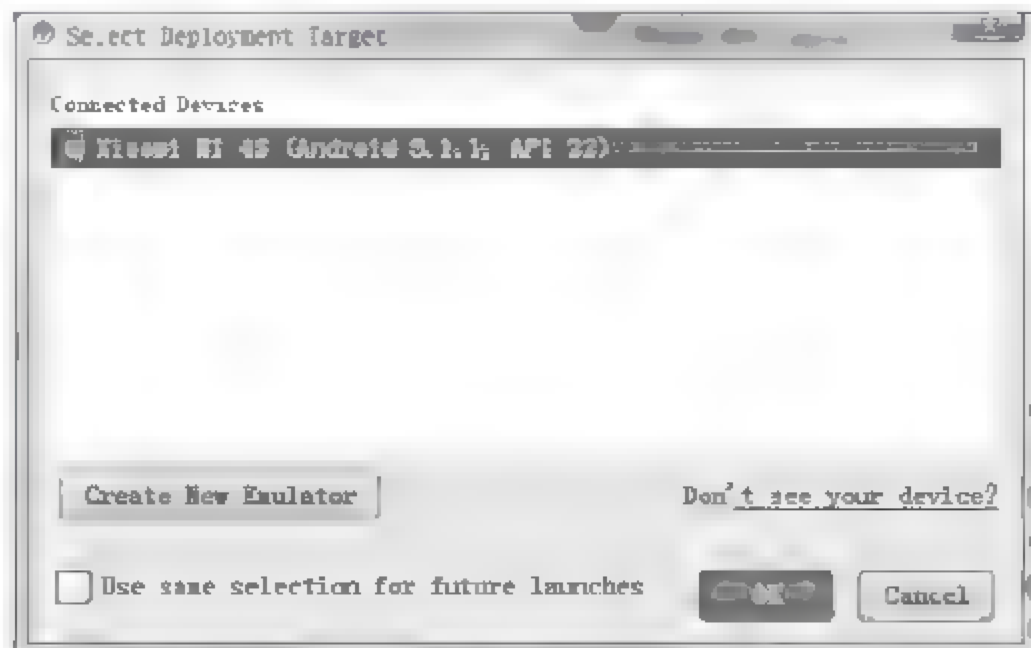


图 8-14　启动 App 时发现真实的手机

8.1.3 导出 APK 安装包

前面说的真机调试是通过数据线把手机连到电脑上，不过在公司的 App 开发工作中，一个 App 要在多部测试手机上安装，难道每次都得把手机拿到手才能安装 App 吗？这么做显然很不方便，此时可以把 App 打包成一个 APK 文件（该文件就是 App 的安装包），然后把 APK 传给测试人员进行后续调试工作。在 Android Studio 中打包 APK，具体步骤如下：

01 依次选择菜单 Build→Generate Signed APK...，弹出窗口如图 8-15 所示。

02 在该窗口中选择待打包的模块名（如 test），单击 Next 按钮，进入 APK 签名窗口页面，如图 8-16 所示。

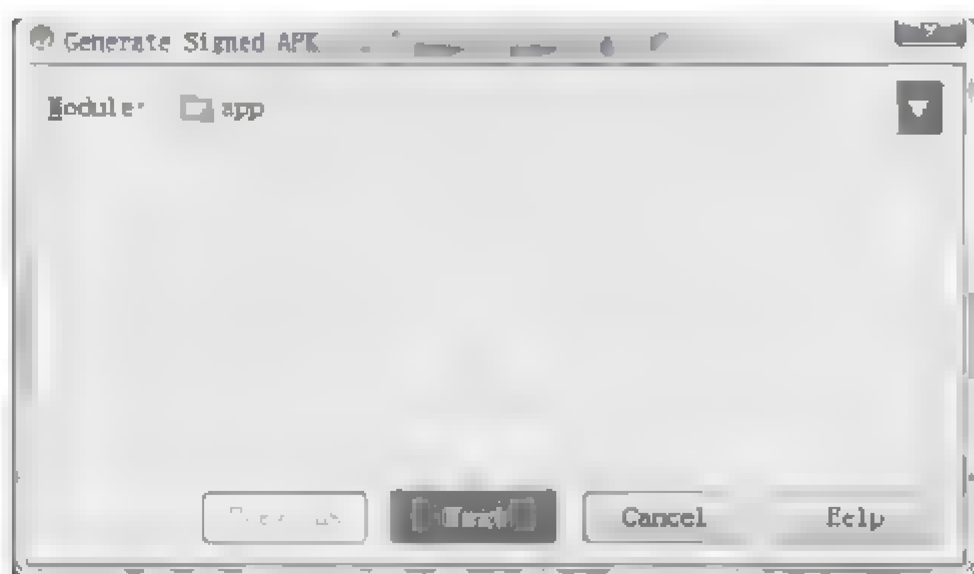


图 8-15 生成安装包的窗口页面

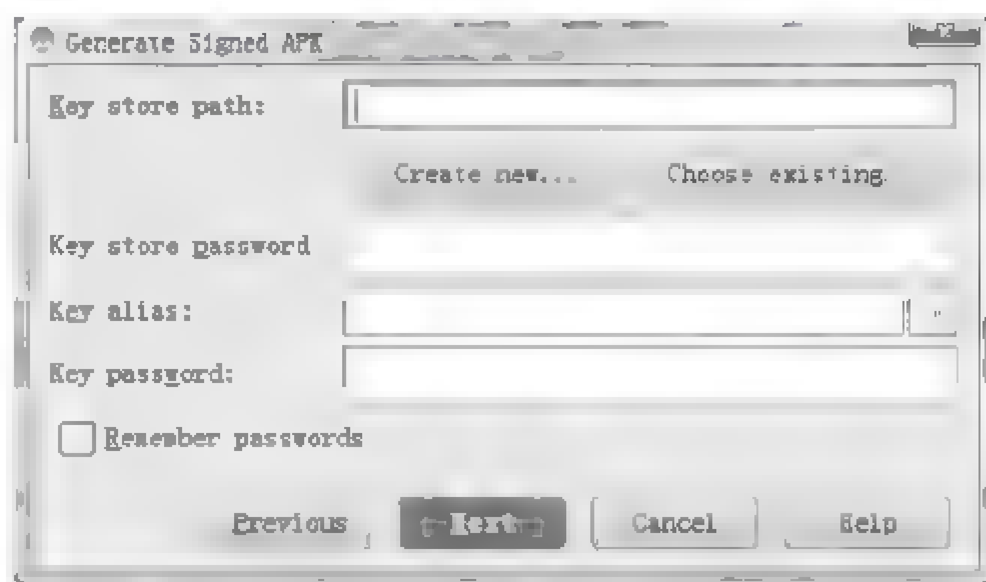


图 8-16 APK 签名的窗口页面

03 在该窗口选择密钥文件的路径，如果原来有密钥文件，就单击 Choose existing...按钮，在弹出的文件对话框中选择密钥文件。如果第一次打包没有密钥文件，就单击 Create new...按钮，然后弹出一个密钥创建窗口，如图 8-17 所示。

04 单击该窗口右上角的 按钮，选择密钥文件的保存路径，单击按钮后弹出文件对话框，如图 8-18 所示。

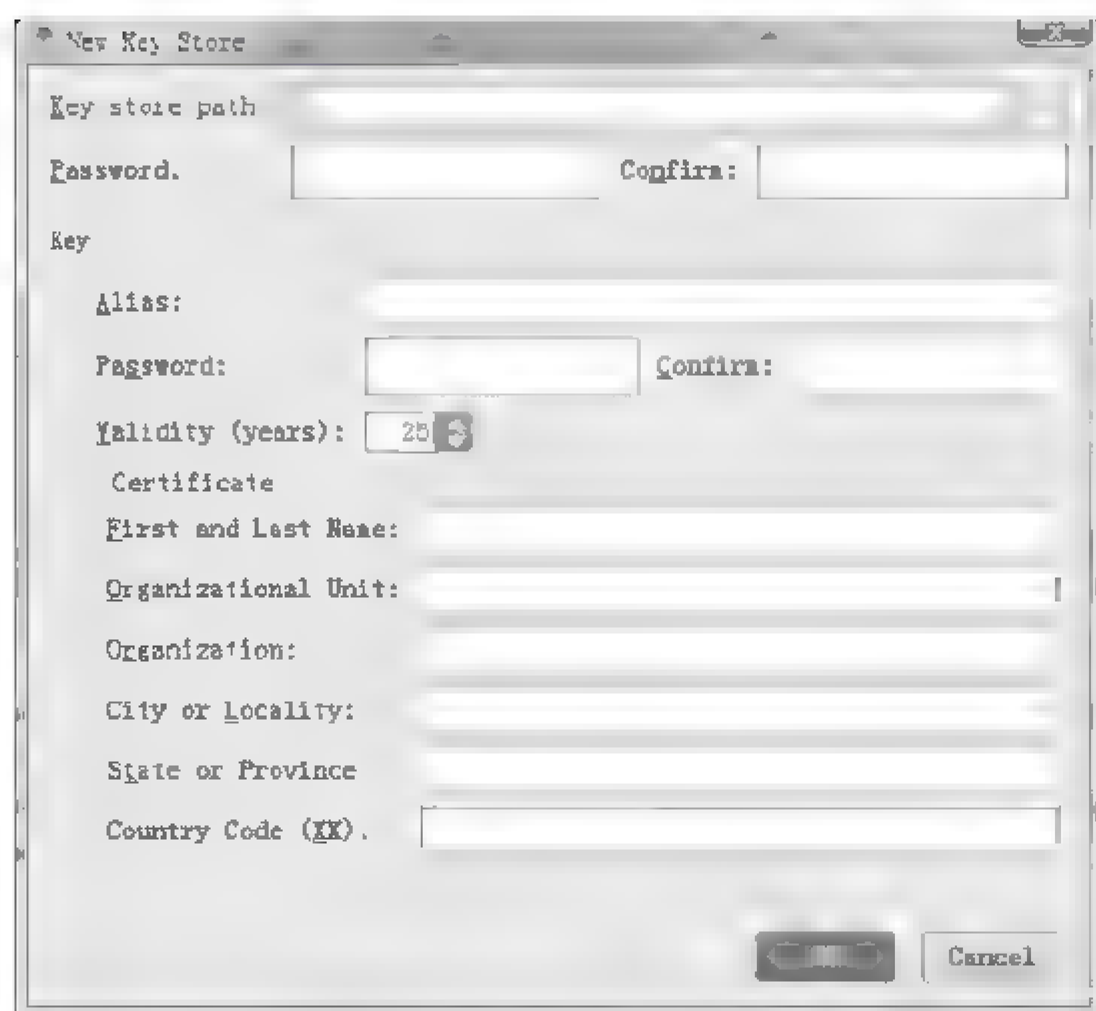


图 8-17 密钥文件的生成窗口

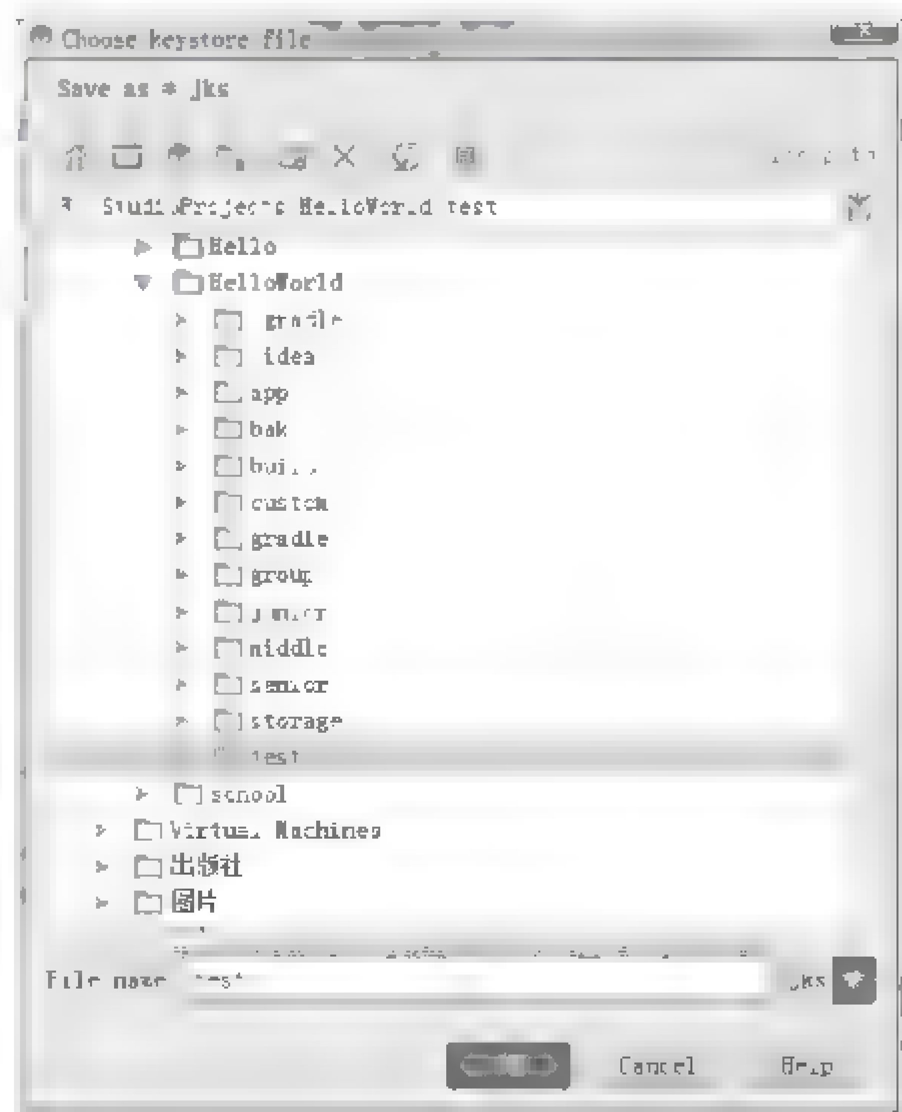


图 8-18 密钥文件的文件对话框

05 在文件对话框中选择文件保存路径，并在下方 File name 右边输入密钥文件的名称，然后单击 OK 按钮回到密钥创建窗口。在该窗口依次填写密码 Password、确认密码 Confirm、别名 Alias、别名密码 Password、别名的确认密码 Confirm，修改密钥文件的有效期限 Validity。下面的输入框只有姓名（First and Last Name）是必填的，填完后的窗口如图 8-19 所示。

06 单击 OK 按钮回到 APK 签名窗口，此时 Android Studio 自动把密码和别名都填上了，如图 8-20 所示。如果一开始选择已存在的密钥文件，这里就要手工输入密码和别名。

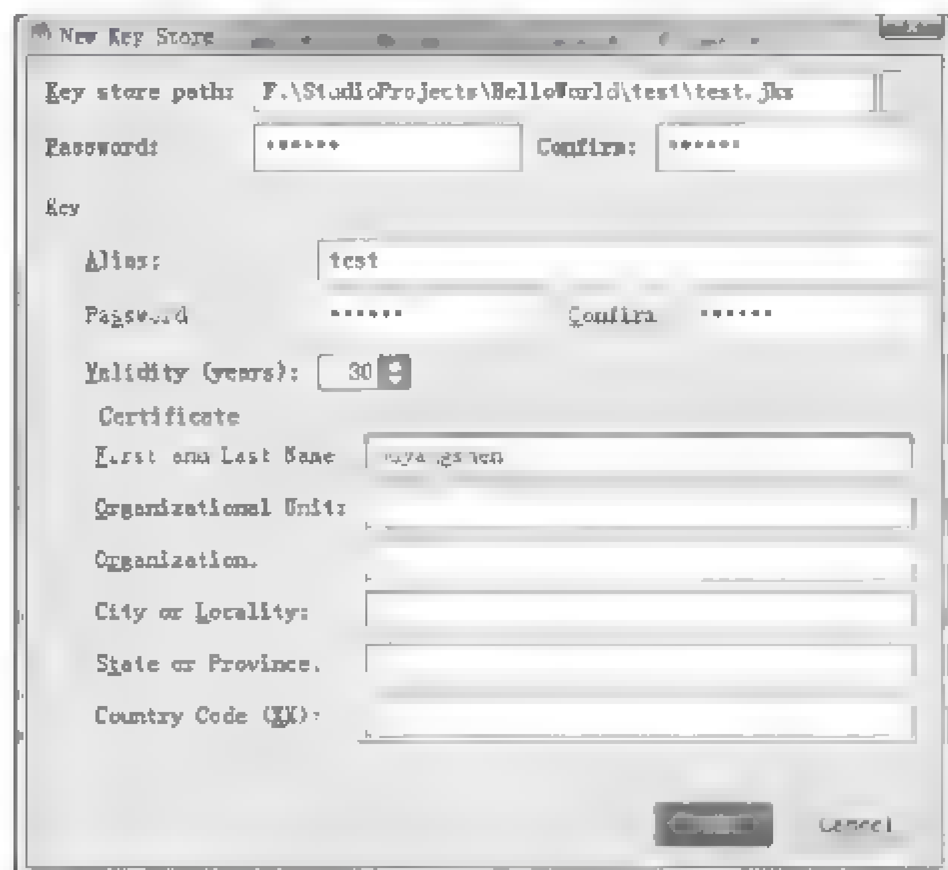


图 8-19　填写完成的密钥创建窗口

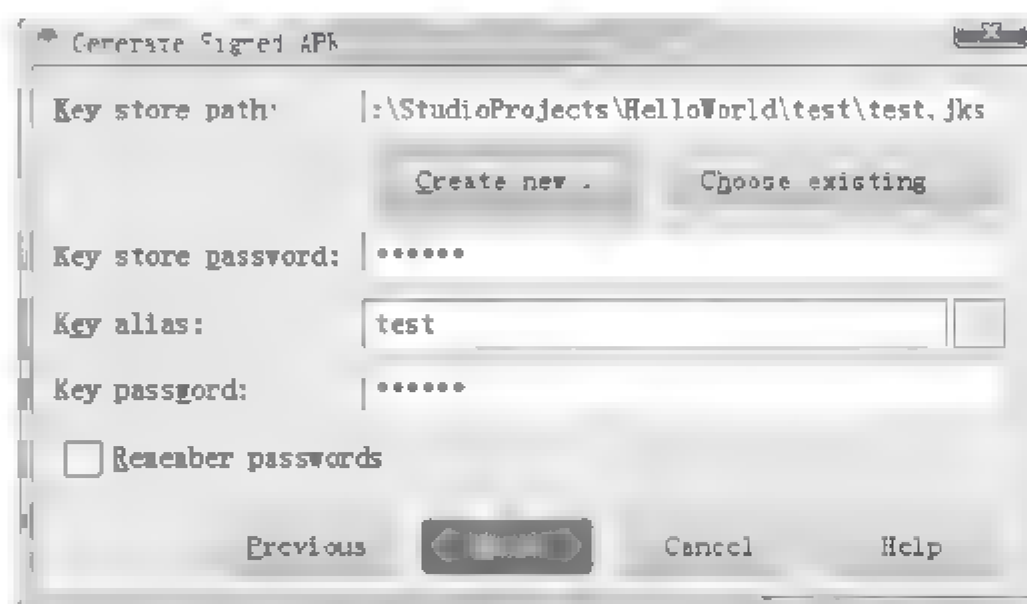


图 8-20　填上签名信息的签名窗口

07 单击 Next 按钮进入下一个页面，如果是启动后第一次打包，就会弹出管理员密码确认窗口，如图 8-21 所示。输入密码再单击 OK 按钮进入 APK 保存页面，如图 8-22 所示。如果不是第一次打包，就直接进入 Apk 保存页面。



图 8-21　管理员密码确认窗口

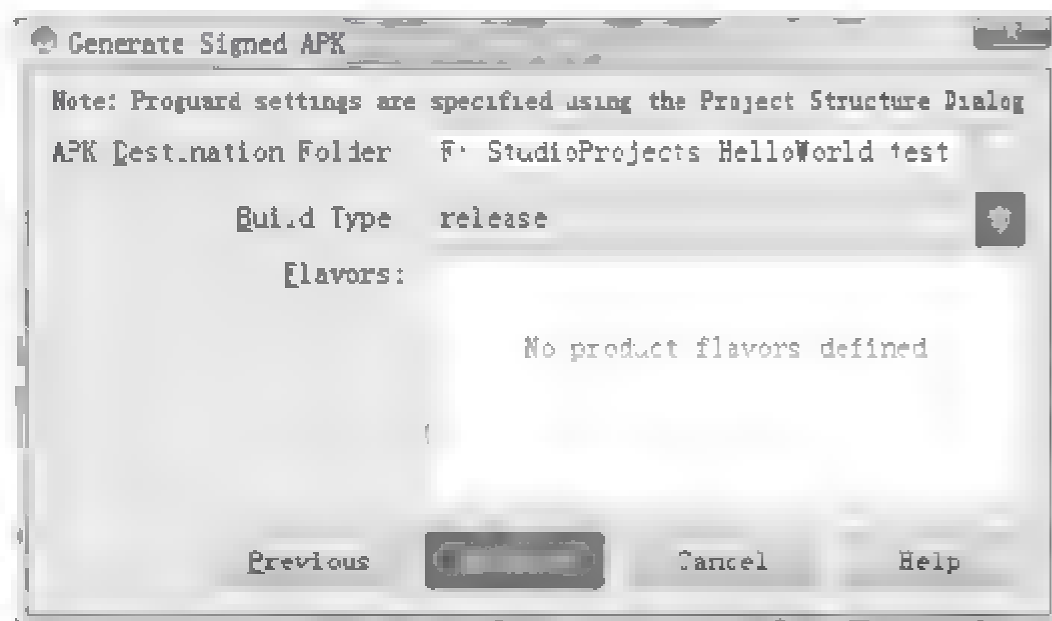


图 8-22　APK 保存页面

APK 保存页面可选择 APK 文件的保存路径和编译类型（Build Type），如果用于调试，编译类型就选择 debug；如果用于发布，编译类型就选择 release。单击 Finish 按钮，等待 Android Studio 生成 APK。

如果没有编译问题，过一会儿就会在 APK 保存路径下看到名为 test-debug.apk 的文件，把该安装包传给其他人，让其他人用数据线把该文件复制到手机的 SD 卡上，然后打开手机的文件管理器，找到这个安装包进行安装。

8.2 准备上线

本节介绍 App 上线前必须做的准备工作，包括正确设置版本信息，例如设置 App 图标、App 名称、App 版本号；把开发模式切换到上线模式，除了代码的切换外，还需修改 AndroidManifest.xml；对关键业务数据进行加密处理，加密算法主要有 MD5、RSA、AES、3DES 等。

8.2.1 版本设置

迄今为止，本书所有演示 App 在屏幕上都显示默认的机器人图标，不过推出一个正式 App 需要用自己设计的图标，而且 App 名称要把英文名换成中文名。App 安装后需要经常升级，所以少不了版本号的管理。开发一个正式 App 需要定制 3 类版本信息，分别是 App 图标、App 名称和 App 版本号。

1. App 图标

App 图标文件是 res/mipmap-***目录下的 ic_launcher.png。若要更改手机桌面上的应用图标，则要把 mipmap-***目录下的 ic_launcher.png 换成新图标。

2. App 名称

App 名称保存在 res/values/strings.xml 的 app_name 中。若要更改手机桌面上的应用名称，则要把 strings.xml 的 app_name 改成新名称。

3. App 版本号

App 版本号放在 build.gradle 的 versionCode 与 versionName 两个参数中。versionCode 必须为整型值，每次升级版本时值都要加 1。versionName 形如“数字.数字.数字”，第一个数字为大版本号，有重要功能升级时，大版本号要加 1，后面两个数字清零；第二个数字为中版本号，每次要进行功能更新时，中版本号加 1，第三个数字清零；第三个数字为小版本号，在有问题修复与界面微调时，小版本号加 1。

注意每次 App 升级，versionCode 与 versionName 都要一起更改，不能只改其中一个。升级后的 versionCode 与 versionName 只能比原来大，不能比原来小。如果没有按照规范修改版本号，就会产生以下问题：

（1）版本号比已安装的版本号小，在安装时直接提示失败，因为 App 只能做升级操作，不能做降级操作。

（2）更新系统内置应用时，如果只修改 versionName，没修改 versionCode，重启手机后就会发现更新丢失，该系统应用已被还原到更新前的版本。这是因为 Android 内核在判断系统应用时会检查 versionCode 的数值，如果 versionCode 不大于当前已安装的版本号，本次更新就被忽略。

下面是获取 App 版本信息的代码片段：

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_version);
    ImageView iv_icon = (ImageView) findViewById(R.id.iv_icon);
    TextView tv_desc = (TextView) findViewById(R.id.tv_desc);
    iv_icon.setImageResource(R.mipmap.ic_launcher);
    try {
        PackageInfo pi = getPackageManager().getPackageInfo(getPackageName(), 0);
        String desc   String.format("App 名称为 ： %s\nApp 版本号为 ： %d\nApp 版本名称为 ： %s",
                getResources().getString(R.string.app_name), pi.versionCode, pi.versionName);
        tv_desc.setText(desc);
    } catch (NameNotFoundException e) {
        e.printStackTrace();
    }
}
```

App 版本信息的获取页面如图 8-23 所示，分别展示了测试 App 的应用图标、应用名称、应用的版本号以及版本名称。

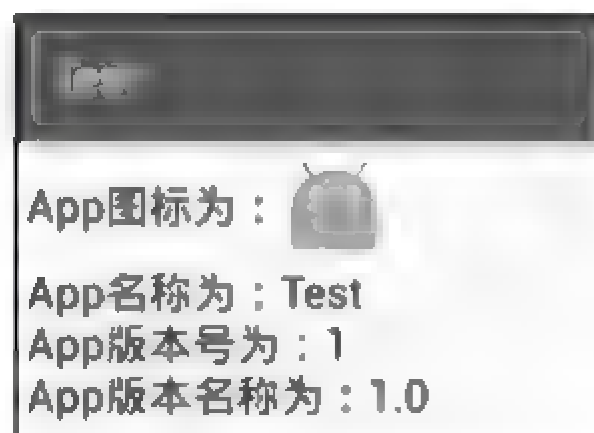


图 8-23　App 版本信息的获取页面

8.2.2　上线模式

为了开发调试方便，程序员常常在代码里添加日志，还在页面上提示各种弹窗。这样固然有利于发现 bug、提高软件质量，不过调试信息过多往往容易泄露敏感信息，例如用户的账号密码、业务流程的逻辑等。从保密需要考虑，App 在上线前需要去掉多余的调试信息，形成上线模式。与之相对的是开发阶段的开发模式。

建立上线模式的好处有以下 3 点：

（1）保护用户的敏感账户信息不被泄露。
（2）保护业务逻辑与流程处理信息不被泄露。
（3）把异常信息转换为更友好的提示信息，改善用户体验。

上线模式不是简单的把调试代码删掉，而是通过某个开关控制是否显示调试信息，因为 App 后续还得修改、更新、重新发布，这个迭代过程要不断调试，从而实现并验证新功能。具体地说，就是建立几个公共类，代码中涉及输入调试信息的地方都改为调用公共类的方法；然后在公共类中定义几个布尔变量作为开关，在开发时打开调试，在上线时关闭调试，从而实现开发模式和上线模式的切换。

控制调试信息的公共类主要有 3 种，分别对 Log 类、Toast 类和 AlertDialog 类进行封装，详细说明如下：

1. Log

Log 类用于打印调试日志。调试 App 时，日志信息会输出到控制台 console 窗口。因为最终用户看不到 App 日志，所以除非特殊情况，发布上线的 App 应屏蔽所有日志信息。

下面是日志工具类的代码：

```
public class LogTool {
    public static boolean isShow = false;   // false 表示上线模式，true 表示开发模式

    public static void v(String tag, String msg) {
        if (isShow == true) {
            Log.v(tag, msg);
        }
    }

    public static void d(String tag, String msg) {
        if (isShow == true) {
            Log.d(tag, msg);
        }
    }

    public static void i(String tag, String msg) {
        if (isShow == true) {
            Log.i(tag, msg);
        }
    }

    public static void w(String tag, String msg) {
        if (isShow == true) {
            Log.w(tag, msg);
        }
    }

    public static void e(String tag, String msg) {
        if (isShow == true) {
            Log.e(tag, msg);
        }
    }

    public static void wtf(String tag, String msg) {
        if (isShow == true) {
            Log.wtf(tag, msg);
        }
    }
}
```

2. Toast

Toast 类用于在界面下方弹出小窗，给用户一两句话的提示，小窗短暂停留一会儿后消失。Toast 窗口无交互动作，样式也基本固定，因此除了少数弹窗可予以保留（如“再按一次返回键退出”），其他弹窗都应在发布时屏蔽。

下面是提示工具类的代码：

```
public class ToastTool {
    public static boolean isShow = false;   // false 表示上线模式，true 表示开发模式

    public static void showShort(Context ctx, String msg) {
        if (isShow == true) {
            Toast.makeText(ctx, msg, Toast.LENGTH_SHORT).show();
        }
    }

    public static void showLong(Context ctx, String msg) {
        if (isShow == true) {
            Toast.makeText(ctx, msg, Toast.LENGTH_LONG).show();
        }
    }

    public static void showQuit(Context ctx) {
        Toast.makeText(ctx, "再按一次返回键退出！", Toast.LENGTH_SHORT).show();
    }
}
```

3. AlertDialog

提醒对话框常用于各种与用户交互的操作，如果是业务逻辑需要，该对话框就无须区分不同模式；如果是提示错误信息，对话框就应该针对两种模式做不同处理。若是开发模式，则对话框展示完整的异常信息，包括输入参数、异常代码、异常描述等；若是上线模式，则对话框展示相对友好的提示文字，如“当前网络连接失败，请检查网络设置是否开启”等。

下面是对话框工具类的代码：

```
public class DialogTool {
    public static boolean isShow = false;   // false 表示上线模式，true 表示开发模式
    public static int SYSTEM = 0;
    public static int IO = 1;
    public static int NETWORK = 2;
    private static String[] mError = { "系统异常，请稍候再试", "读写失败，请清理内存空间后再试",
            "网络连接失败，请检查网络设置是否开启" };

    public static void showError(Context ctx, int type, String title, Exception e) {
        AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
```

```
        builder.setTitle(title);
        if (isShow == true) {
            String desc = String.format("%s\n 异常描述 ： %s", mError[type], e.getMessage());
            builder.setMessage(desc);
        } else {
            builder.setMessage(mError[type]);
        }
        builder.setPositiveButton("确定", null);
        builder.create().show();
    }

    public static void showError(Context ctx, int type, String title, int code, String msg) {
        AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
        builder.setTitle(title);
        if (isShow == true) {
            String desc = String.format("%s\n 异常代码：%d\n 异常描述：%s", mError[type], code, msg);
            builder.setMessage(desc);
        } else {
            builder.setMessage(mError[type]);
        }
        builder.setPositiveButton("确定", null);
        builder.create().show();
    }
}
```

除了代码外，AndroidManifest.xml 还要区分开发模式与上线模式，有以下 3 点修改说明。

（1）application 标签中加上属性 android:debuggable="true"表示调试模式，默认 false 表示上线模式。若在模拟器上调试或通过 Android Studio 直接把 App 安装到手机上，则无论 debuggable 的值是多少都直接切换到调试模式。在上线发布时要把该属性设置为 false。

（2）App 发布后，没有特殊情况，开发者都不希望 activity 和 service 对外开放。但其默认是对外部开放的，所以要在 activity 和 service 标签下分别添加属性 android:exported= "false"，表示该组件不对外开放。

（3）App 默认安装到内部存储，因为手机与平板的存储空间有限，所以应该尽量让 App 选择安装到 SD 卡，避免占用宝贵的内部存储空间。这时要在 manifest 标签下加上属性 android:installLocation，该属性的取值说明见表 8-1。

表 8-1　安装位置的取值说明

安装位置的类型	说明
internalOnly	默认值，只能安装在内部存储。无法通过安全软件的应用搬家功能将其挪到 SD 卡
auto	优先装在内部存储，但若内部存储空间不足，则会安装在 SD 卡。安装之后，用户可通过安全软件选择是否将其挪到 SD 卡。推荐设为该值
preferExternal	安装在 SD 卡上。但若 SD 卡不存在或 SD 卡空间不足，则安装在内部存储



8.2.3　数据加密

大家都知道，数据安全很重要，现在无论干什么都要密码，各种账号和密码一旦泄露必将造成财产损失。但是 Android 对数据安全的支持很弱，并没有很好的数据保密措施。

例如，共享参数 SharedPreferences 本质上是操作一个 XML 配置文件，文件具体路径为/data/data/应用包名/shared_prefs/***.xml；打开 shared.xml 共享参数文件后里面全部都是明文：

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="name">Mr Lee</string>
    <int name="age" value="30" />
    <boolean name="married" value="true" />
    <float name="weight" value="100.0" />
</map>
```

如果里面存放用户的银行账号与密码，不要说是黑客，就是一个 App 初学者拿到别人手机后也一样容易获得其中的用户账号信息。

SQLite 数据库也不安全，数据库文件具体路径为/data/data/应用包名/databases/***.db。这个 db 文件未经加密处理，只要弄来 sqlitemanager 等 SQLite 的管理工具，就能轻松查看数据库中存储的各种信息。图 8-24 所示为使用 sqlitemanager 查看某 App 数据库的表记录。

Table: NavItem

id	value	label	logo	updateTime	parentId
1	2	教育阅读	app/appLogo/2.png	2016-07-06 16:56:35	-1
2	3	生活助手	app/appLogo/3.png	2016-07-06 16:57:07	-1
3	4	亲子乐园	app/appLogo/4.png	2016-07-06 16:57:24	-1
4	1	游戏	app/appLogo/1.png	2016-07-06 16:57:54	-1
5	201	认知教学	app/appLogo/201.png	2016-08-02 17:03:13	2
6	202	英语学习	app/appLogo/202.png	2016-08-02 17:03:29	2
7	203	课堂辅导	app/appLogo/203.png	2016-08-02 17:03:45	2
8	204	儿童绘本	app/appLogo/204.png	2016-08-02 17:03:55	2

图 8-24　SQLite 保存的数据记录信息

用户数据无论保存在 SharedPreferences，还是保存在 SQLite 数据库，都有必要对关键数据进行加密。加密算法多种多样，常见的有 MD5、RSA、AES、3DES 四种。

1. MD5 加密

MD5 是不可逆的加密算法，也就是无法解密，主要用于客户端的用户密码加密。MD5 算法的加密代码如下：

```
public class MD5Util {
    public static String encryp(String raw) {
        String md5Str = raw;
        try {
            MessageDigest md = MessageDigest.getInstance("MD5");
            md.update(raw.getBytes());
```

```
                byte[] encryContext = md.digest();
                int i;
                StringBuffer buf = new StringBuffer("");
                for (int offset = 0; offset < encryContext.length; offset++) {
                    i = encryContext[offset];
                    if (i < 0) {
                        i += 256;
                    }
                    if (i < 16) {
                        buf.append("0");
                    }
                    buf.append(Integer.toHexString(i));
                }
                md5Str = buf.toString();
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
            return md5Str;
        }
    }
```

MD5 算法的加密效果如图 8-25 所示，无论原始字符串是什么，MD5 加密串都是 32 位的十六进制字符串。



图 8-25　MD5 算法的加密结果

2. RSA 加密

RSA 算法在客户端使用公钥加密，在服务端使用私钥解密。这样一来，即使加密的公钥被泄露，没有私钥仍然无法解密。

下面是 RSA 加密的 3 个注意事项。

（1）需要导入加密算法的依赖包 bcprov-jdk16-1.46.jar，该 jar 包要放在当前模块的 libs 目录下。

（2）RSA 加密的结果是字节数组，经过 BASE64 编码才能形成最终的加密字符串。

（3）依据需求要对加密前的字符串做 reverse 倒序处理。

RSA 算法的加密代码如下：

```
public class RSAUtil {
    private static final String Algorithm = "RSA";

    private static byte[] encrypt(PublicKey pk, byte[] data) throws Exception {   //加密
        try {
            Cipher cipher = Cipher.getInstance(Algorithm,
                    new org.bouncycastle.jce.provider.BouncyCastleProvider());
```

```
            cipher.init(Cipher.ENCRYPT_MODE, pk);
            int blockSize = cipher.getBlockSize();
            int outputSize = cipher.getOutputSize(data.length);
            int leavedSize = data.length % blockSize;
            int blocksSize = leavedSize != 0 ? data.length / blockSize + 1 : data.length / blockSize;
            byte[] raw = new byte[outputSize * blocksSize];
            int i = 0;
            while (data.length - i * blockSize > 0) {
                if (data.length - i * blockSize > blockSize) {
                    cipher.doFinal(data, i * blockSize, blockSize, raw, i * outputSize);
                } else {
                    cipher.doFinal(data, i * blockSize, data.length - i * blockSize, raw, i * outputSize);
                }
                i++;
            }
            return raw;
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
    }

    private static byte[] decrypt(PrivateKey pk, byte[] raw) throws Exception {    //解密
        try {
            Cipher cipher = Cipher.getInstance(Algorithm,
                        new org.bouncycastle.jce.provider.BouncyCastleProvider());
            cipher.init(cipher.DECRYPT_MODE, pk);
            int blockSize = cipher.getBlockSize();
            ByteArrayOutputStream bout = new ByteArrayOutputStream(64);
            int j = 0;
            while (raw.length - j * blockSize > 0) {
                bout.write(cipher.doFinal(raw, j * blockSize, blockSize));
                j++;
            }
            return bout.toByteArray();
        } catch (Exception e) {
            throw new Exception(e.getMessage());
        }
    }

    //使用 N、e 值还原公钥
    private static PublicKey getPublicKey(String modulus, String publicExponent, int radix)
                throws NoSuchAlgorithmException, InvalidKeySpecException {
        BigInteger bigIntModulus = new BigInteger(modulus, radix);
```

```
        BigInteger bigIntPrivateExponent = new BigInteger(publicExponent, radix);
        RSAPublicKeySpec keySpec = new RSAPublicKeySpec(bigIntModulus, bigIntPrivateExponent);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PublicKey publicKey = keyFactory.generatePublic(keySpec);
        return publicKey;
    }

    // 使用 N、d 值还原私钥
    private static PrivateKey getPrivateKey(String modulus, String privateExponent, int radix)
            throws NoSuchAlgorithmException, InvalidKeySpecException {
        BigInteger bigIntModulus = new BigInteger(modulus, radix);
        BigInteger bigIntPrivateExponent = new BigInteger(privateExponent, radix);
        RSAPrivateKeySpec keySpec = new RSAPrivateKeySpec(bigIntModulus, bigIntPrivateExponent);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PrivateKey privateKey = keyFactory.generatePrivate(keySpec);
        return privateKey;
    }

    public static String encodeRSA(RSAKeyData key_data, String src) {   //加密函数
        if (key_data == null) {   //以下为测试用的密钥对
            key_data = new RSAKeyData();
            key_data.public_key = "10001";
            key_data.private_key = "";
            key_data.modulus = "c7f668eccc579bb75527424c21be31c104bb44c921b4788ebc82cddab
5042909eaea2dd70643153l392d79890f9091e13714285a7e79e9d1836397f847046ef2519c9b65022b48bf157fe409
f8a42155734e65467d04ac844dfa0c2ae512517102986ba9b62d67d4c920eae40b2f11c363b218a703467d342faa8171
9f57e2c3";
            key_data.radix = 16;
        }
        try {
            PublicKey key = getPublicKey(key_data.modulus, key_data.public_key, key_data.radix);
            String rev = encodeURL(new StringBuilder(src).reverse().toString());
            byte[] en_byte = encrypt(key, rev.getBytes());
            String base64 = encodeURL(ConvertBytesToBase64.BytesToBase64String(en_byte));
            return base64;
        } catch (Exception e) {
            e.printStackTrace();
            return "RSA 加密失败";
        }
    }

    private static String encodeURL(String str) {   //URL 编码
        String encode_str = str;
```

```
        try {
            encode_str = URLEncoder.encode(str, "utf-8");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return encode_str;
    }

    private static String decodeURL(String str) {   //URL 解码
        String decode_str = str;
        try {
            decode_str = URLDecoder.decode(str, "utf-8");
        } catch (Exception e) {
            e.printStackTrace();
        }
        return decode_str;
    }

    public static class RSAKeyData {
        public String modulus;
        public String public_key;
        public String private_key;
        public int radix;

        public RSAKeyData() {
            modulus = "";
            public_key = "";
            private_key = "";
            radix = 0;
        }
    }
}
```

RSA 算法的加密效果如图 8-26 所示，加密结果是经过 URL 编码的字符串。



图 8-26　RSA 算法的加密结果

3. AES 加密

AES 是设计用来替换 DES 的高级加密算法。下面是该算法加密和解密的代码：

```
public class AesUtil {
    private static final String Algorithm = "AES";
    private final static String HEX = "0123456789ABCDEF";

    //加密函数。key 为密钥
    public static String encrypt(String key, String src) throws Exception {
        byte[] rawKey = getRawKey(key.getBytes());
        byte[] result = encrypt(rawKey, src.getBytes());
        return toHex(result);
    }

    //解密函数。key 值必须和加密时的 key 一致
    public static String decrypt(String key, String encrypted) throws Exception {
        byte[] rawKey = getRawKey(key.getBytes());
        byte[] enc = toByte(encrypted);
        byte[] result = decrypt(rawKey, enc);
        return new String(result);
    }

    private static void appendHex(StringBuffer sb, byte b) {
        sb.append(HEX.charAt((b >> 4) & 0x0f)).append(HEX.charAt(b & 0x0f));
    }

    private static byte[] getRawKey(byte[] seed) throws Exception {
        KeyGenerator kgen = KeyGenerator.getInstance(Algorithm);
        // SHA1PRNG 强随机种子算法，要区别 Android 4.2.2 以上版本的调用方法
        SecureRandom sr = null;
        if (android.os.Build.VERSION.SDK_INT >= 17) {
            sr = SecureRandom.getInstance("SHA1PRNG", "Crypto");
        } else {
            sr = SecureRandom.getInstance("SHA1PRNG");
        }
        sr.setSeed(seed);
        kgen.init(256, sr);  // 256 位/128 位/192 位
        SecretKey skey = kgen.generateKey();
        byte[] raw = skey.getEncoded();
        return raw;
    }

    private static byte[] encrypt(byte[] key, byte[] src) throws Exception {
        SecretKeySpec skeySpec = new SecretKeySpec(key, Algorithm);
        Cipher cipher = Cipher.getInstance(Algorithm);
```

```
            cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
            byte[] encrypted = cipher.doFinal(src);
            return encrypted;
        }

        private static byte[] decrypt(byte[] key, byte[] encrypted) throws Exception {
            SecretKeySpec skeySpec = new SecretKeySpec(key, Algorithm);
            Cipher cipher = Cipher.getInstance(Algorithm);
            cipher.init(Cipher.DECRYPT_MODE, skeySpec);
            byte[] decrypted = cipher.doFinal(encrypted);
            return decrypted;
        }

        private static byte[] toByte(String hexString) {
            int len = hexString.length() / 2;
            byte[] result = new byte[len];
            for (int i = 0; i < len; i++) {
                result[i] = Integer.valueOf(hexString.substring(2 * i, 2 * i + 2), 16).byteValue();
            }
            return result;
        }

        private static String toHex(byte[] buf) {
            if (buf == null) {
                return "";
            }
            StringBuffer result = new StringBuffer(2 * buf.length);
            for (int i = 0; i < buf.length; i++) {
                appendHex(result, buf[i]);
            }
            return result.toString();
        }
    }
```

AES 算法的加密效果如图 8-27 所示。该算法是可逆算法，支持对加密字符串进行解密，前提是解密时密钥必须与加密时一致。



图 8-27　AES 算法的加密结果

4. 3DES 加密

3DES（Triple DES）是三重数据加密算法，相当于对每个数据块应用 3 次 DES 加密算法。因为原先 DES 算法的密钥长度过短，容易遭到暴力破解，所以 3DES 算法通过增加密钥的长度防范加密数据被破解。在实际开发中，3DES 的密钥必须是 24 位的字节数组，过短或过长在运行时都会报错 java.security.InvalidKeyException。另外，3DES 加密生成的是字节数组，也得通过 BASE64 编码为文本形式的加密字符串。

该算法的加密和解密代码如下：

```
public class Des3Util {
    private static final String Algorithm = "DESede";  // 定义加密算法 DESede，即 3DES

    public static String encrypt(String key, String raw) {  //加密函数。key 为密钥
        byte[] enBytes = encryptMode(key, raw.getBytes());
        BASE64Encoder encoder = new BASE64Encoder();
        return encoder.encode(enBytes);
    }

    public static String decrypt(String key, String enc) {  //解密函数。key 值必须和加密时的 key 一致
        try {
            BASE64Decoder decoder = new BASE64Decoder();
            byte[] enBytes = decoder.decodeBuffer(enc);
            byte[] deBytes = decryptMode(key, enBytes);
            return new String(deBytes);
        } catch (IOException e) {
            e.printStackTrace();
        return enc;
        }
    }

    private static byte[] encryptMode(String key, byte[] src) {
        try {
            SecretKey deskey = new SecretKeySpec(build3DesKey(key), Algorithm);
            Cipher cipher = Cipher.getInstance(Algorithm);
            cipher.init(Cipher.ENCRYPT_MODE, deskey);
            return cipher.doFinal(src);
        }catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    private static byte[] decryptMode(String key, byte[] src) {
        try {
            SecretKey deskey = new SecretKeySpec(build3DesKey(key), Algorithm);
```

```
                Cipher cipher = Cipher.getInstance(Algorithm);
                cipher.init(Cipher.DECRYPT_MODE, deskey);
                return cipher.doFinal(src);
            }catch (Exception e) {
                e.printStackTrace();
                return null;
            }
        }

        //根据字符串生成密钥 24 位的字节数组
        private static byte[] build3DesKey(String keyStr) throws UnsupportedEncodingException {
            byte[] key = new byte[24];
            byte[] temp = keyStr.getBytes("UTF-8");
            if (key.length > temp.length) {
                System.arraycopy(temp, 0, key, 0, temp.length);
            }else {
                System.arraycopy(temp, 0, key, 0, key.length);
            }
            return key;
        }
    }
```

3DES 算法的加密效果如图 8-28 所示。该算法与 AES 一样是可逆算法，支持对加密字符串进行解密，前提是解密时密钥必须与加密时一致。



图 8-28　3DES 算法的加密结果

8.3　安全加固

本节介绍对 APK 安装包进行安全加固的过程。首先通过反编译工具成功破解 App 源码，从而表明对 APK 实施安全防护的必要性和紧迫性；接着详细说明代码混淆的原理与规则，演示代码混淆如何加大源码破译的难度；然后描述怎样利用第三方加固网站对 APK 做加固处理；最后演示对加固包进行重签名的方法。

8.3.1　反编译

编译是把代码编译为程序，反编译是把程序破解为代码。



谁都不想自己的劳动成果被别人窃取，何况是辛辛苦苦敲出来的 App 代码，然而由于 Java 语言的特性，Java 写的程序往往容易被反编译破解，只要获得 App 的安装包，就能通过反编译工具破解出该 App 的完整源码。开发者绞尽脑汁上架一个 App，结果这个 App 却被他人从界面到代码都“山寨”了，那可真是欲哭无泪了。为了说明代码安全的重要性，下面对反编译的完整过程进行介绍，警醒开发者防火、防盗、防破解。

首先准备反编译的 3 个工具，分别是 apktool、dex2jar、jd-gui，注意下载最新版本。

- apktool：对 APK 文件进行解包，主要用来解析 res 资源和 AndroidManifest.xml。
- dex2jar：将 APK 包中的 classes.dex 转为 JAR 包，该 JAR 包就是 App 代码的编译文件。
- jd-gui：将 dex2jar 解析出来的 jar 包反编译为 Java 源码。

下面是反编译 APK 的具体步骤（以 Window 环境举例说明）。

步骤 01 打开 DOS 命令窗口，进入 apktool 所在的目录，运行“apktool.bat d -f 解包后的保存目录名 待处理的 APK 文件名”，等待反编译过程，如图 8-29 所示。

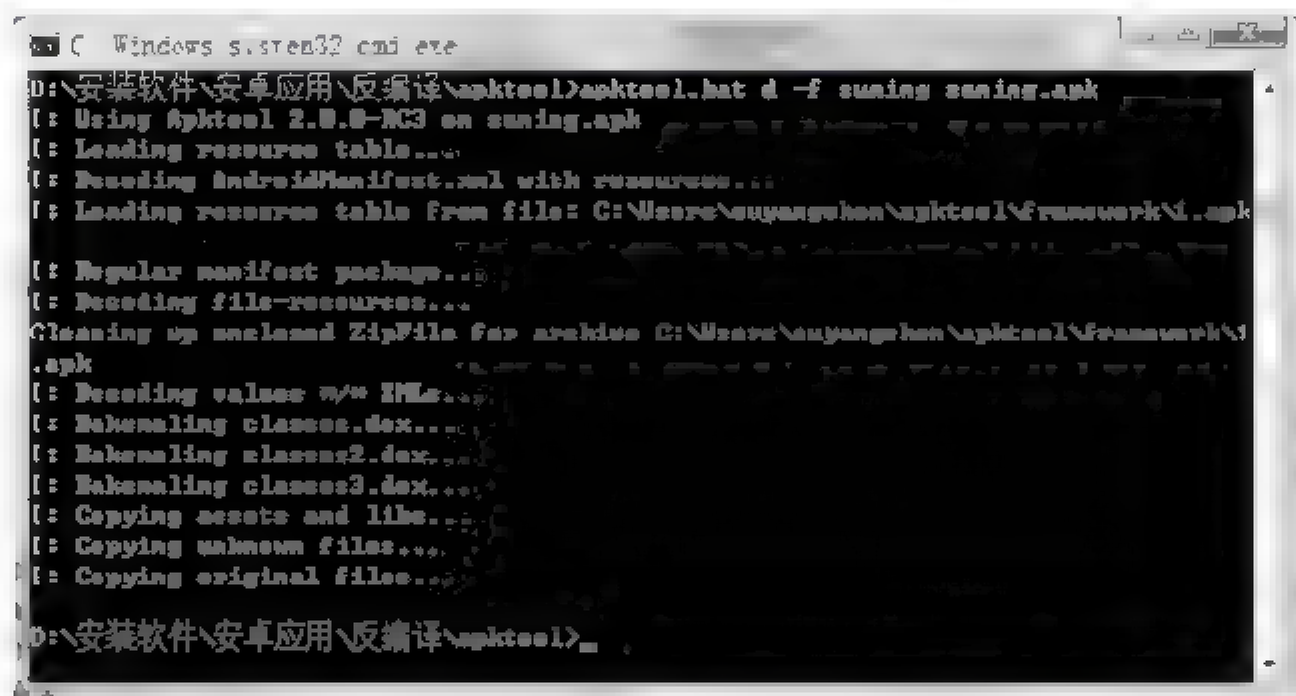


图 8-29　反编译工具 apktool 的运行截图

反编译通过，即可在当前目录下看到破解目录。apktool 的主要目的是解析出 res 资源，包括 AndroidManifest.xml 和 res/layout、res/values、res/drawable 等目录下的资源文件。

步骤 02 用压缩软件（如 Winrar）打开 APK 包，APK 安装包其实是一个压缩文件，使用 Winrar 打开 APK 文件的目录结构如图 8-30 所示。



图 8-30　Apk 解压后的内部目录结构

先从 APK 包中解压出 classes.dex 文件，再进入 dex2jar 所在的目录，运行命令 d2j-dex2jar.bat classes.dex，等到破解完成，即可在当前目录下看到新文件 classes_dex2jar.jar，该 JAR 包即为 App 源码的编译文件。

步骤 03 双击打开 jd-gui.exe，用鼠标把 classes_dex2jar.jar 拖到 jd-gui 界面中，程序就会自动把 JAR 包反编译为 Java 源码，反编译后的 Java 源码目录结构如图 8-31 所示。

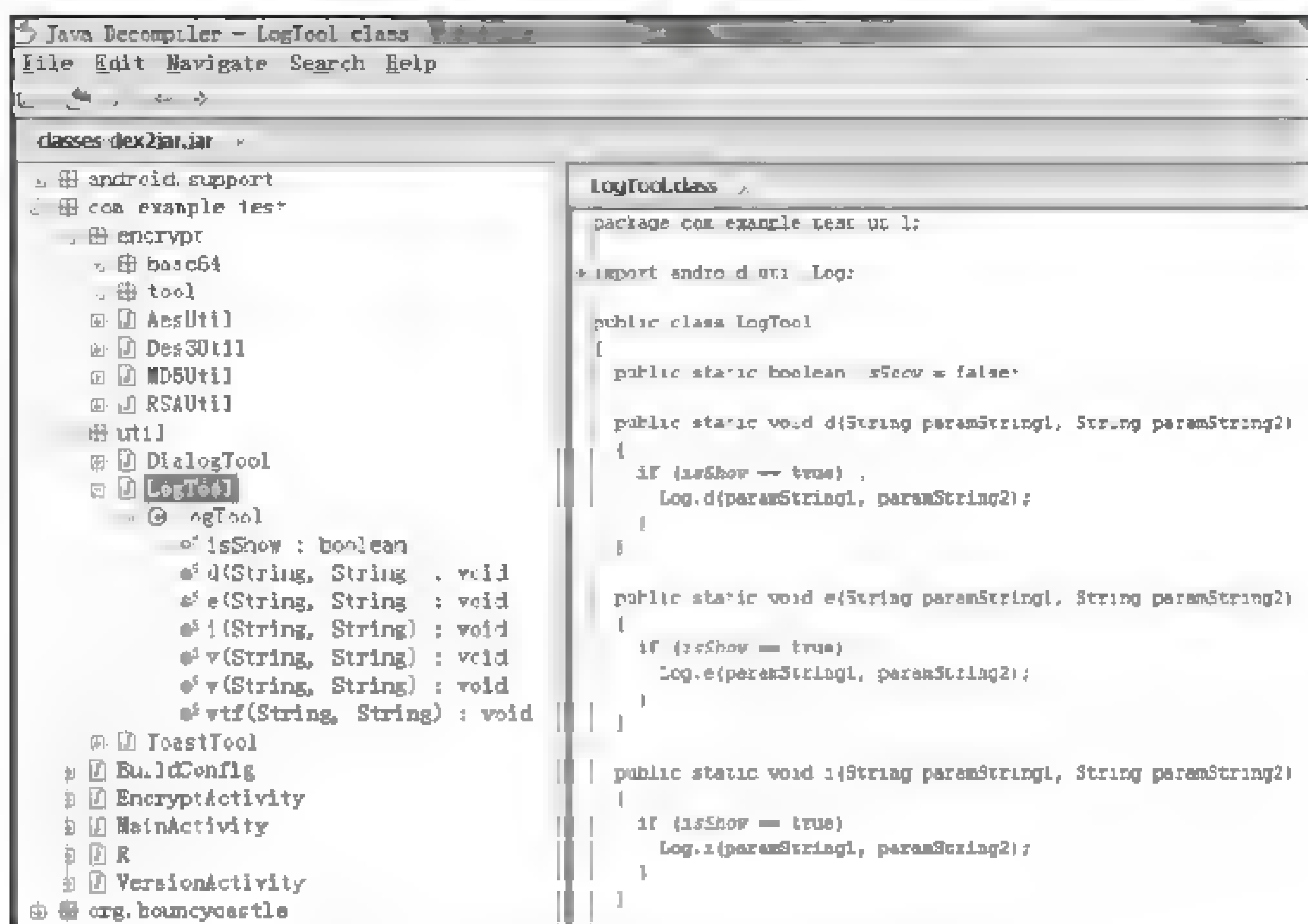


图 8-31　反编译后的 java 源码目录结构

在 jd-gui 界面依次选择菜单 File→Save All Sources，输入保存路径，在指定目录生成 zip 文件，解压 zip 文件就能看到反编译后的全部 Java 代码了。

上面的反编译过程不但破解了 Java 代码，而且 res 资源文件也被一起破解了，如果你的 App 不采取一些保护措施，整个工程源码就会暴露在大庭广众之下。所以要赶紧扯些遮羞布盖上，穿好衣服裤子，告别原始社会，走进文明社会。

8.3.2　代码混淆

前面说到反编译能够破解 App 的整个工程源码，因此有必要对 App 源码采取防护措施，代码混淆就是保护代码安全的措施之一。Android Studio 已经自带了代码混淆器 ProGuard，用途包括以下两点：

（1）压缩 APK 包的大小，删除无用代码，并简化部分类名和方法名。

（2）加大破解源码的难度，部分类名和方法名被重命名使得程序逻辑变得难以理解。

代码混淆的配置文件其实一直都存在，只是我们之前都将其忽略了。每次在 Android Studio 新建一个模块，该模块的根目录下都会自动生成 proguard-rules.pro。打开 build.gradle，在 android→buildTypes→release 节点下可以看到两行编译配置：

```
minifyEnabled false
proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
```

Android Studio 默认不做代码混淆，上面第一行的 minifyEnabled 为 false 表示关闭混淆功能，要把该参数改为 true 才能开启混淆功能。第二行配置指定 proguard-rules.pro 作为本模块的代码混淆文件，该文件保存的是各种详细的代码混淆规则。

下面是 proguard-rules.pro 的一个模板：

```
#指定代码的压缩级别
-optimizationpasses 5
#是否使用大小写混合
-dontusemixedcaseclassnames
#优化/不优化输入的类文件
-dontoptimize
#是否混淆第三方 JAR 包
-dontskipnonpubliclibraryclasses
#混淆时是否做预校验
-dontpreverify
#混淆时是否记录日志
-verbose
#混淆时所采用的算法
-optimizations !code/simplification/arithmetic,!field/*,!class/merging/*
#保护注解
-keepattributes *Annotation*
#保持 JNI 用到的 native 方法不被混淆
-keepclasseswithmembers class * {
    native <methods>;
}
#保持自定义控件的构造函数不被混淆，因为自定义控件很可能直接写在布局文件中
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet);
}
#保持自定义控件的构造函数不被混淆
-keepclasseswithmembers class * {
    public <init>(android.content.Context, android.util.AttributeSet, int);
}
#保持布局中 onClick 属性指定的方法不被混淆
-keepclassmembers class * extends android.app.Activity {
   public void *(android.view.View);
}
#保持枚举 enum 类不被混淆
-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}
#保持序列化的 Parcelable 不被混淆
```

```
-keep class * implements android.os.Parcelable {
  public static final android.os.Parcelable$Creator *;
}
#指定哪些第三方 JAR 包需要混淆
#-libraryjars libs/bcprov-jdk16-1.46.jar
#保持哪些系统组件类不被混淆
-keep public class * extends android.app.Fragment
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
-keep public class * extends android.support.v4.**
-keep public class com.android.vending.licensing.ILicensingService
#保持哪些第三方 JAR 包不被混淆。比如上一节 RSA 算法用到了 bcprov-jdk16-1.46.jar，该 JAR 包里的工具类就不可混淆
-keep class org.bouncycastle.**
-dontwarn org.bouncycastle.**
```

进行代码混淆时有以下 5 点注意事项：

（1）对某些特殊的类或方法屏蔽混淆，可能会在布局文件中直接引用类名或方法名，包括自定义控件、布局中 onClick 属性指定的方法等。

（2）保持第三方 JAR 包不被混淆，有时需要把 keep class 提到 dontwarn 前面。

（3）JAR 包的文件名中不要有特殊字符，比如“(”“)”等字符在混淆时会报错，文件名最好只包含字母、横线、小数点。

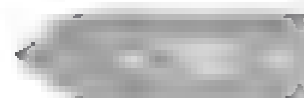
（4）jni 的方法要屏蔽混淆，因为 so 库要求包名、类名、函数名完全一致。

（5）使用 WebView 时会被 js 调用的类和方法也要屏蔽混淆。具体做法除了要在 proguard-rules.pro 加上说明外，还要在 Java 代码中调用 js 使用的方法，才能保证内部类与相关方法都没有被混淆。

```
-keep class com.example.mixture.WebActivity$MobileSignal{
     public <fields>;
     public <methods>;
}
```

经过代码混淆后重新生成的 APK 文件，再用反编译工具进行破解，反编译后的 Java 源码结构如图 8-32 所示。

从图中看到，混淆处理后的包名与类名都变成了 a、b、c、d 这样的名称，无疑加大了黑客理解源码的难度。试想当黑客面对这些天书般的 a、b、c、d，还会想要绞尽脑汁地尝试破译吗？



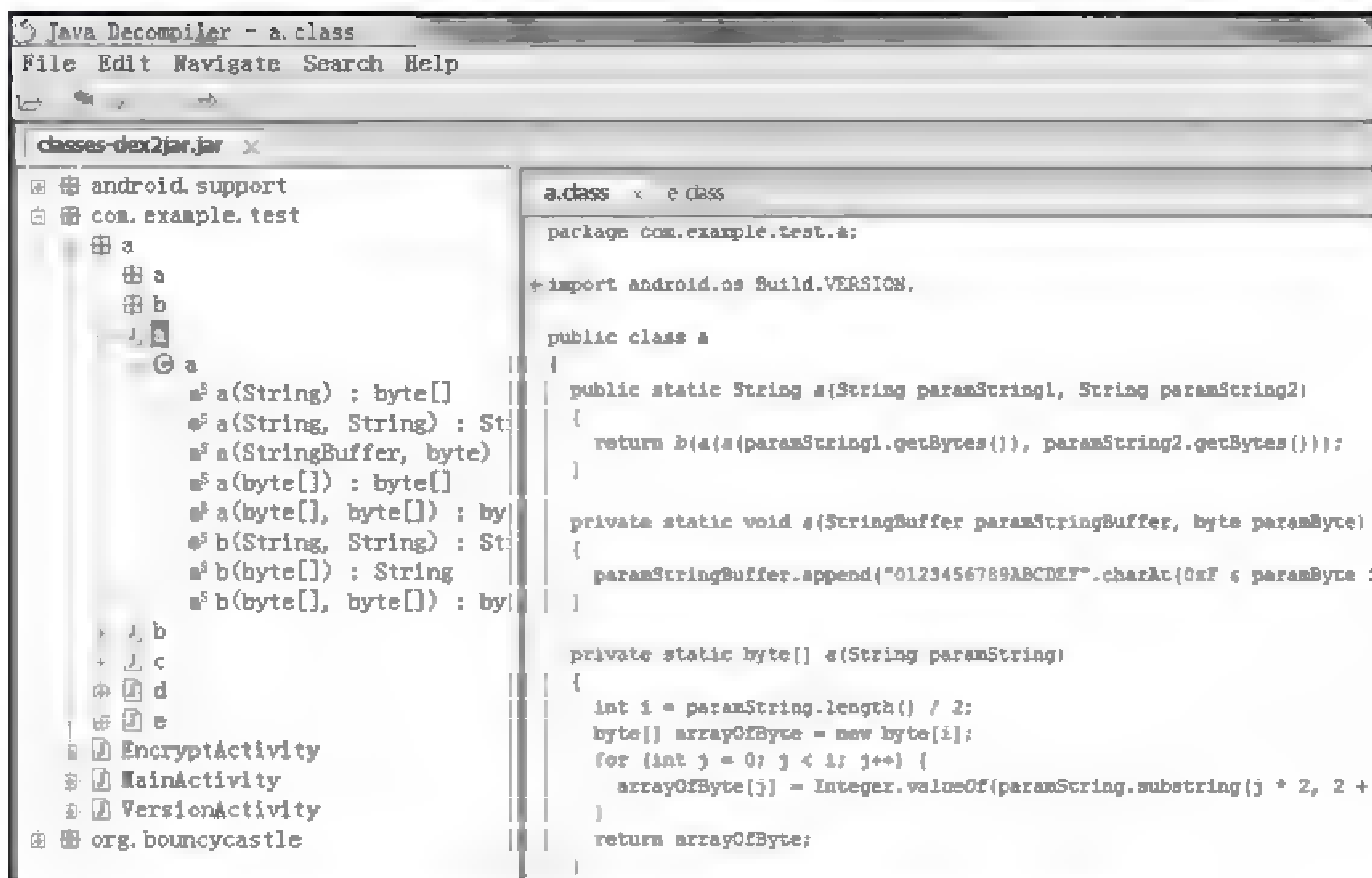


图 8-32　经过代码混淆再破解后的 Java 源码目录结构

8.3.3　第三方加固及重签名

App 经过代码混淆后初步结束了裸奔的状态，但代码混淆只能加大源码破译的难度，并不能完全阻止被破解。除了代码破解外，App 还存在其他安全风险，比如二次打包、篡改内存、漏洞暴露等情况。对于这些安全风险，Android Studio 基本无能为力。因此，鉴于术业有专攻，我们不如把 APK 文件交给专业加固网站进行加固处理。举个做得比较好的第三方加固的例子，360 加固保的网址是 http://jiagu.360.cn/。开发者要先在该网站注册新用户，然后打开在线加固页面，加固页面如图 8-33 所示。

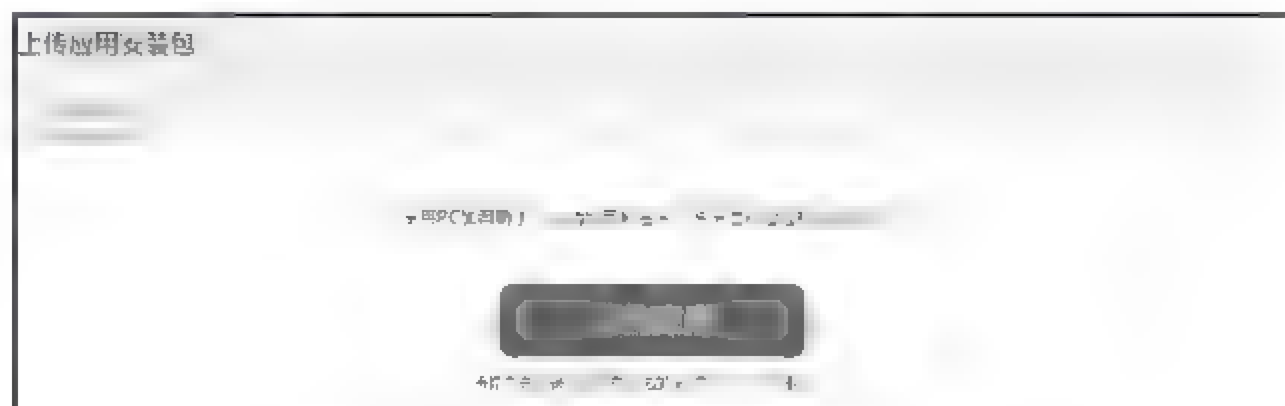


图 8-33　360 加固保的在线加固页面

单击该页面的“上传应用”按钮，上传成功后跳到下一页，向下拉到页面底部，选中“正版签名”开启加固按钮，如图 8-34 所示。



图 8-34　确认加固页面

单击“开始加固”按钮，跳到应用信息页面，如图 8-35 所示。



图 8-35 加固后的应用信息页面

应用信息中部的当前状态为“加固成功”，单击右边的“下载应用”按钮，把加固好的安装包下载到本地，下载后的文件名如 test-release.encrypted.apk。此时用反编译工具尝试破解这个加固包，会发现该安装包变得无法破译。

加固后的 APK 破坏了原来的签名，无法直接安装到手机上，所以要对该文件进行重签名，才能成为合法的 APK 安装包。重签名用到两个工具，分别是 jarsigner 和 zipalign，具体说明如下：

1. jarsigner

jarsigner 是 Java 自带的 JAR 包签名工具，路径为 Java 安装目录下的 jdk/bin/jarsigner.exe，使用命令的格式为“jarsigner -verbose -keystore 密钥文件全路径 -storepass 密钥文件的密码 -keypass 别名的密码 -digestalg SHA1 -sigalg MD5withRSA -signedjar 签名后的文件名 待签名的文件名 别名”。

2. zipalign

zipalign 是 Android 开发工具包 SDK 自带的 APK 优化工具，相当于内存对齐从而提高读取效率，路径为 SDK 安装目录下的 build-tools/版本号/zipalign.exe，使用命令的格式为“zipalign -v 4 已签名的文件名 对齐后的文件名”。

下面对加固好的 APK 文件进行重签名，完整的命令如下：

```
jarsigner -verbose -keystore test.jks -storepass 111111 -keypass 111111 -digestalg SHA1 -sigalg
MD5withRSA -signedjar test-release-signed.apk test-release.encrypted.apk test
zipalign -v 4 test-release-signed.apk test-release-signed-align.apk
```

上述命令里的 test-release-signed.apk 表示签名后的文件，test-release-signed-align.apk 表示对齐后的最终安装包。

当然，命令行方式不够友好，现在有专门的重签名软件，比如爱加密的签名软件 APKSign，下载该软件并安装，安装完毕后打开 APKSign，该软件的界面如图 8-36 所示。

在 APKSign 界面上选择待签名的 APK，再选择签名文件的路径，然后依次输入密码、别名、别名的密码、签名后的存放路径，输入效果如图 8-37 所示，最后单击“开始签名”按钮完成签名操作。

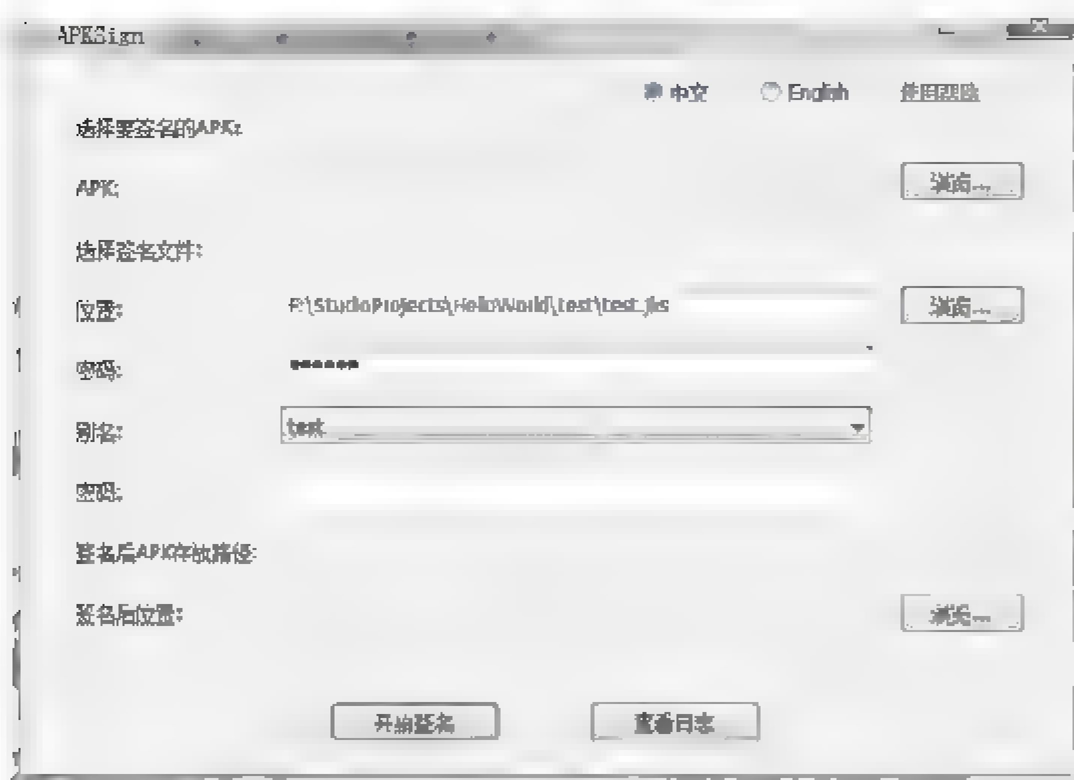


图 8-36　爱加密的重签名工具界面

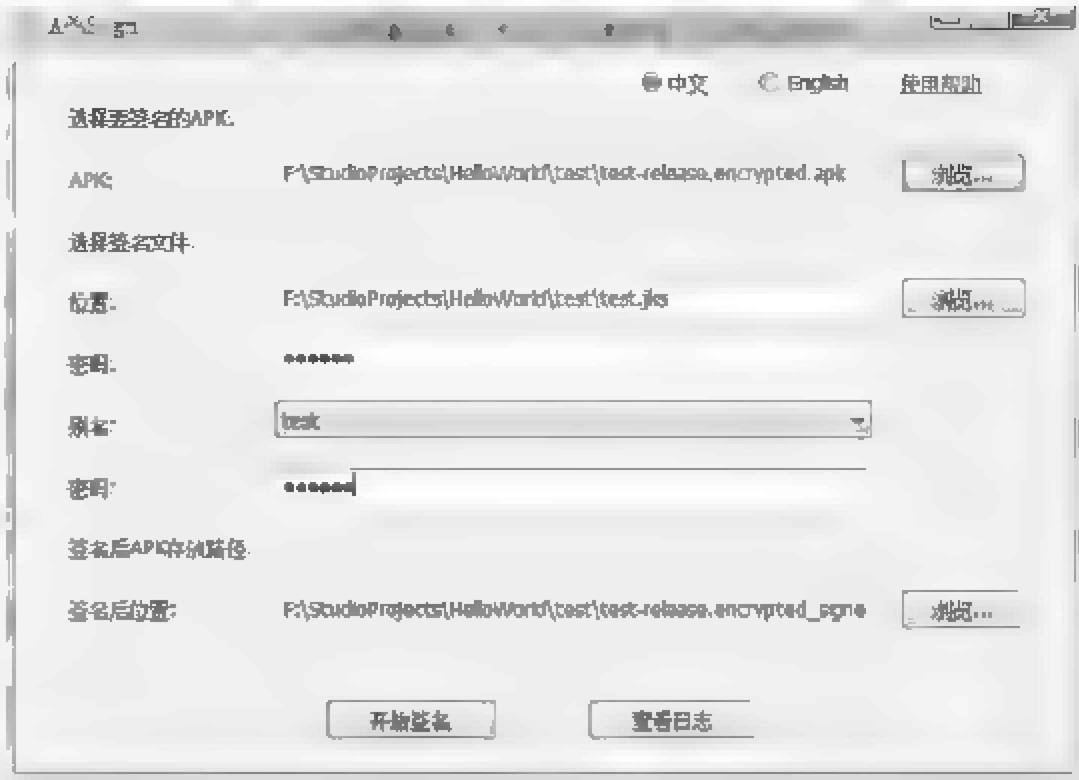


图 8-37　信息填写好的重签名工具界面

8.4　发布到应用商店

本节介绍把 App 发布到应用商店的过程。首先要在应用商店注册开发者账号，以腾讯开放平台为例说明开发者注册账号的步骤；然后使用已注册的开发者账号在开放平台上创建并提交应用；最后描述如何查看应用上线的审核结果，以应用宝 App 为例说明搜索并安装已上线 App 的方法。

8.4.1　注册开发者账号

APK 文件完成签名后，可谓是万事俱备，只欠东风了，接下来把 App 发布到各大应用市场。主要的应用商店有应用宝、百度手机助手、360 手机助手、小米应用商店、华为应用商店、豌豆荚等。下面举例说明如何将你的 App 发布到应用宝。

应用宝只是手机上的应用商店 App 名称，对应的开发者后台是腾讯开放平台，网址是 http://op.open.qq.com/。读者应该都有 QQ 账号，直接用 QQ 号码登录腾讯开放平台，跳转到开发者注册页面，如图 8-38 所示。

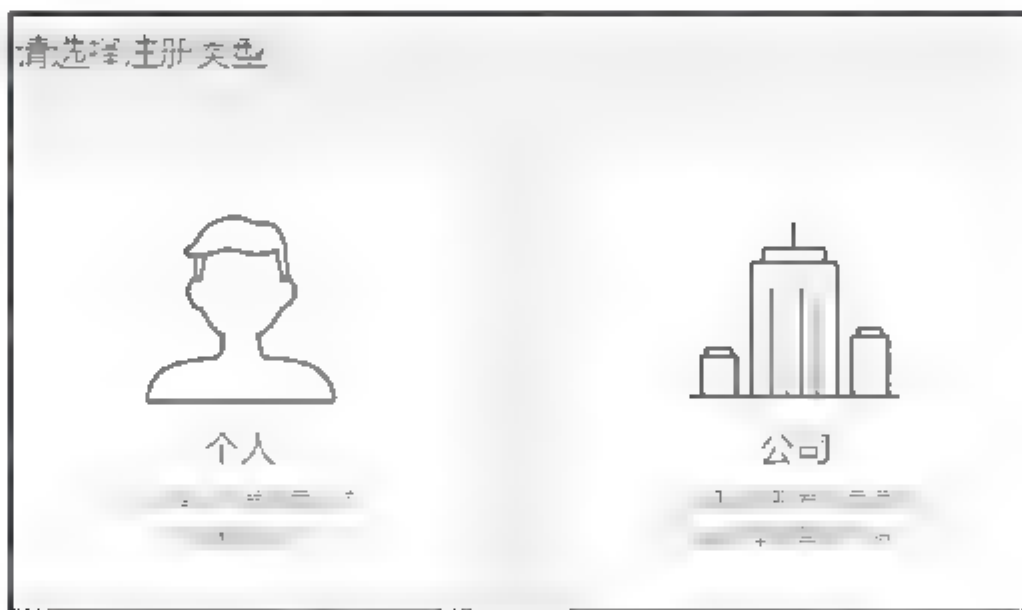


图 8-38　腾讯开放平台的开发者注册页面

如果是个人开发者，就单击左边的“个人”类型；如果是公司开发者，就单击右边的“公司”类型。这里我们选择“个人”类型，跳转到下一页的个人资料页面填写个人信息，如图 8-39 所示；填写联系方式，如图 8-40 所示。

个人资料填写完成后，单击“下一步”按钮，跳转到验证邮箱页面。打开你的注册邮箱，找到腾讯开放平台的开发者注册认证邮件，点击邮件中的确认链接，完成开发者的注册认证。

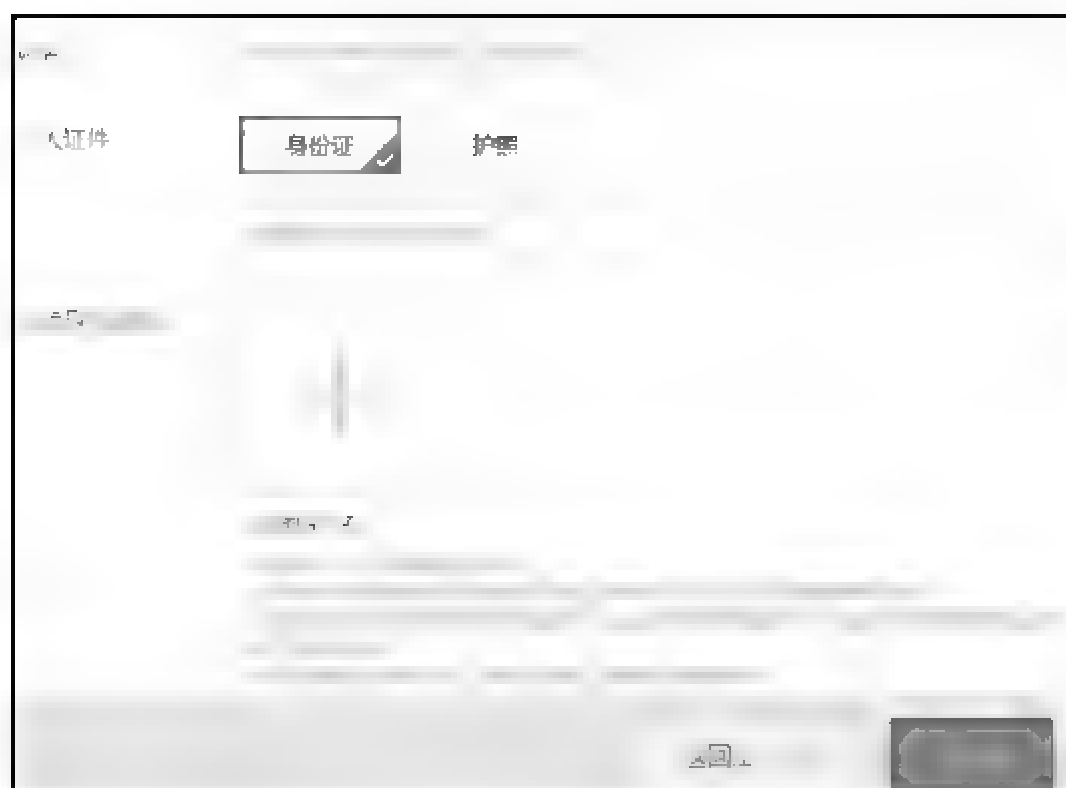


图 8-39　个人信息的填写页面

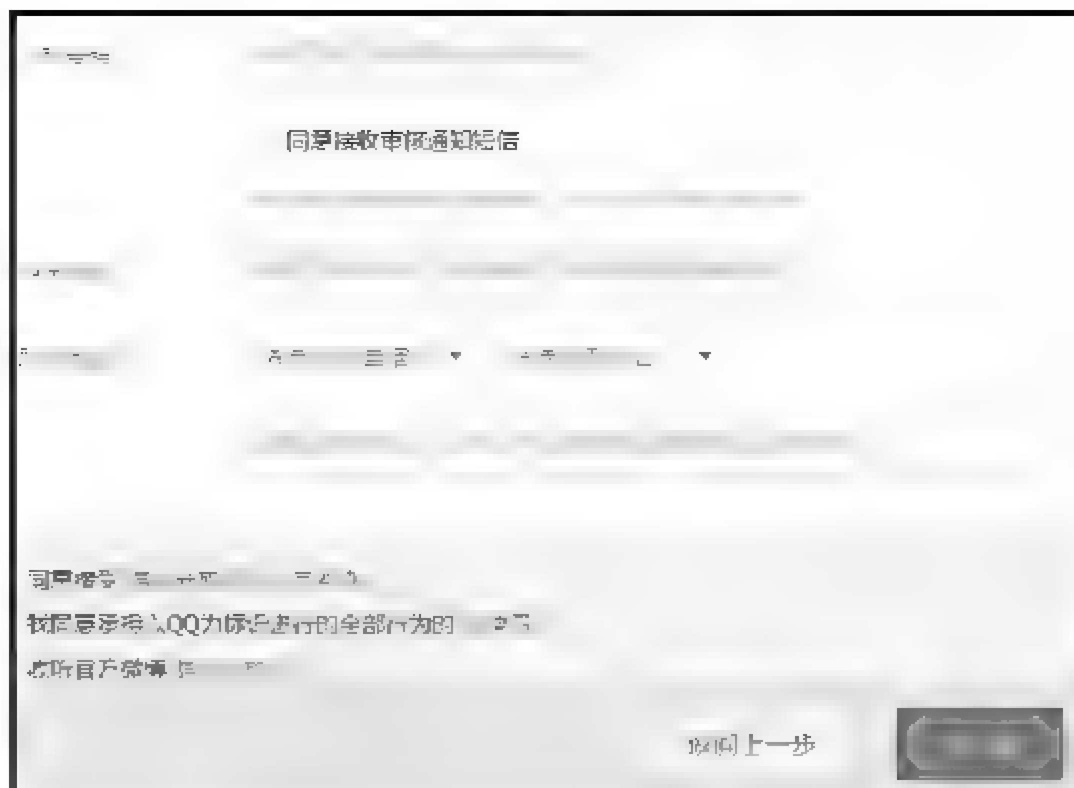


图 8-40　联系方式的填写页面

8.4.2　创建并提交应用

开发者注册完成后，回到腾讯开放平台的主页，在管理中心页面上单击“创建应用”按钮，跳转到应用创建页面，如图 8-41 所示。



图 8-41　腾讯开放平台的应用创建页面

在该页面上选择最左边的“移动应用 安卓”，然后单击下方的“创建应用”按钮，在弹出的类型对话框中选择“软件”，如图 8-42 所示。

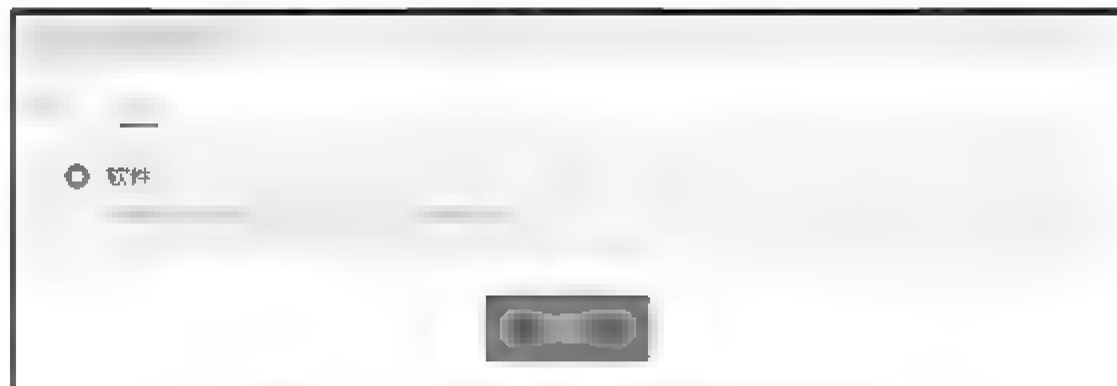


图 8-42　应用类型的选择对话框

单击“确定”按钮，跳转到应用信息填写页面，依次填写应用的各项基本信息，包括应用名称、应用类型、应用标签、应用简介等，示例效果如图 8-43 所示。

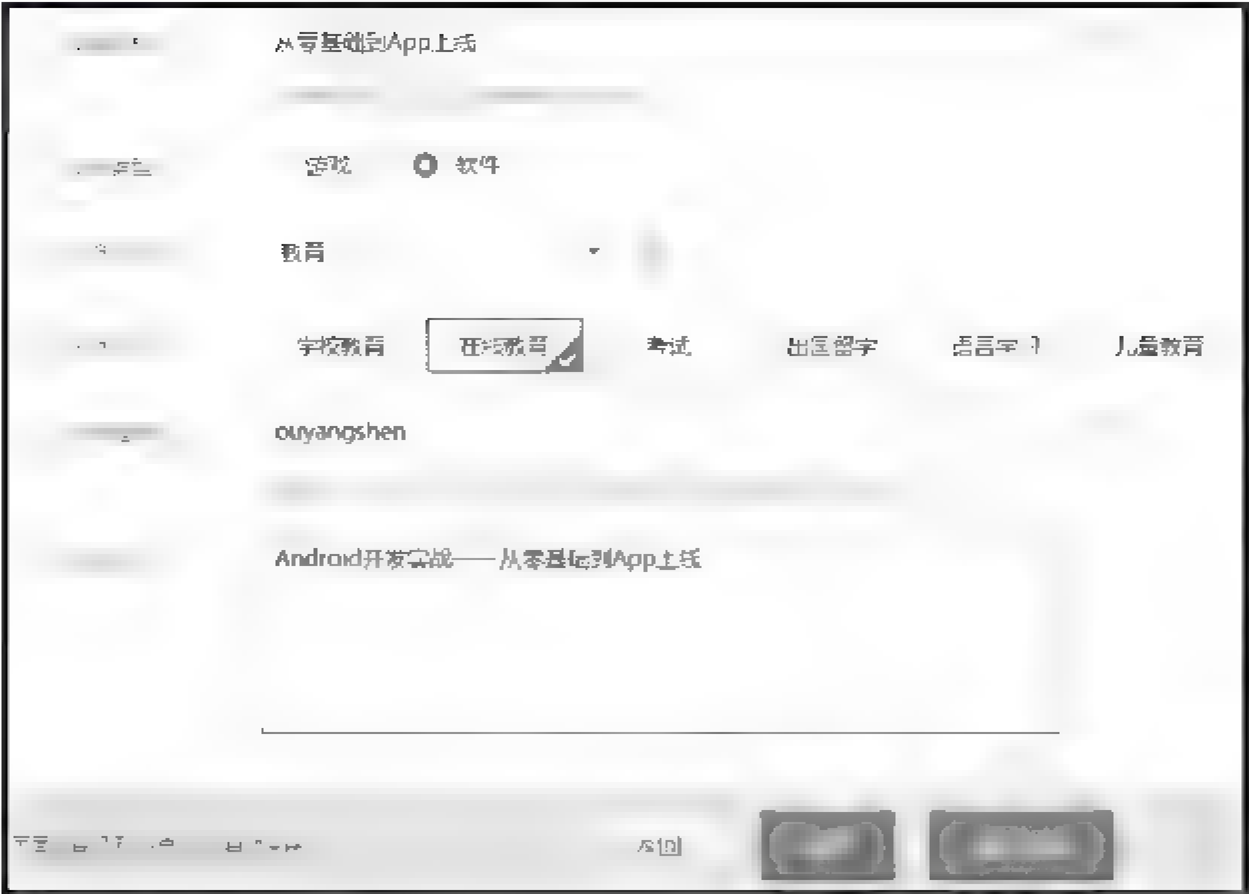


图 8-43　应用信息的填写页面

分别上传安装包、应用图标，填写应用的适配信息，如图 8-44 所示。



图 8-44　上传安装包、应用图标等信息的页面

最后单击页面右下方的“提交审核”按钮，等待开放平台的人工审核。审核一般需要 1～3 个工作日，一旦通过审核，你的 QQ 邮箱会收到一封审核通知邮件。此时重新打开腾讯开放平台，进入管理中心，页面上多了一个已上线应用的记录，如图 8-45 所示。

已上线(1)　未上线(2)　请输入完整的应用名称或ID　创建应用

应用名称	状态	上线时间	应用等级	日下载量	总下载量	操作
商城助手	已上线	[illegible]	C	0	87	更新安装包

图 8-45　管理中心的已上线应用记录

若想验证该 App 是否确实上线成功，则可打开手机上的应用宝 App，搜索该 App 的名称，搜索结果会出现该 App 的应用信息，如图 8-46 所示。





图 8-46 应用宝 App 上的应用搜索结果

点击该应用右边的“下载”按钮，即可开始下载操作，下载完毕后点击“安装”按钮，App 就可以成功安装到用户手机上。

8.5 小 结

本章主要介绍了 App 从调试到发布的详细过程，包括调试工作（模拟器调试、真机调试、导出 APK 安装包）、准备上线（版本设置、上线模式、数据加密）、安全加固（反编译、代码混淆、第三方加固及重签名）、发布到应用商店（注册开发者账号、创建并提交应用、从应用商店下载应用）。经过这一系列应用发布流程，完成了 App 从开发阶段的代码到用户手机上的应用的华丽转变，实现 App“开发”→“测试”→“加固”→“上线”的完整过程。

通过本章的学习，读者应该能够掌握以下 5 种开发技能：

（1）学会通过模拟器和真机对 App 进行调试。
（2）学会把 App 工程从开发模式转为上线模式。
（3）学会利用签名证书导出 APK 安装包。
（4）学会对 APK 包进行安全加固和重签名。
（5）学会把 App 发布到各大应用商店。



设备操作

本章介绍 App 开发常用的一些设备操作，主要包括如何使用摄像头进行拍照、如何使用麦克风进行录音并结合摄像头进行录像、如何播放录制好的音频和视频、如何使用常见传感器实现业务功能、如何使用定位功能获取位置信息等。最后结合本章所学的知识演示一个实战项目“仿微信的发现功能”的设计与实现。

9.1 摄 像 头

本节介绍利用摄像头实现相机功能的办法，首先对表面视图SurfaceView的用法进行说明，演示如何运用相机类Camera结合表面视图完成拍照功能（含单拍和连拍）。然后对表面视图的升级版——纹理视图TextureView的用法进行阐述，并演示如何在新版Camera2架构中结合纹理视图完成拍照功能（含单拍和连拍）。

9.1.1 表面视图SurfaceView

Android的绘图机制是由UI线程在屏幕上绘图，一般情况下不允许其他线程直接做绘图操作。这个机制在处理简单页面时没什么问题，因为普通页面不会频繁且大面积地绘图，但是该机制在处理复杂多变的页面时会产生问题，比如时刻变化着的游戏界面、拍照或录像时不断变换着的预览界面就会导致UI线程资源堵塞，即界面卡死的状况。

表面视图SurfaceView是Android用来解决子线程绘图的特殊视图，拥有独立的绘图表面，即不与其宿主页面共享同一个绘图表面。由于拥有独立的绘图表面，因此表面视图的界面能够在一个独立线程中进行绘制，这个子线程为渲染线程。因为渲染线程不占用主线程资源，所以一方面可以实现复杂而高效的UI刷新，另一方面及时响应用户的输入事件。由于表面视图具备以上特性，因此可用于拍照和录像的预览界面，也可用于游戏的实时界面。

因为表面视图不在UI主线程绘图，无论是onDraw方法还是dispatchDraw方法都没有进行绘图操作，所以表面视图必然要通过其他途径绘图，这个途径便是内部类表面持有者SurfaceHolder外部调用SurfaceView对象的getHolder方法获得SurfaceHolder对象，然后进行预览界面的相关绘图操作。

下面是SurfaceHolder的常用方法。

- lockCanvas：锁定并获取绘图表面的画布。
- unlockCanvasAndPost：解锁并刷新绘图表面的画布。
- addCallback：添加绘图表面的回调接口SurfaceHolder.Callback。回调接口有以下3个方法。
  - surfaceCreated：在绘图表面创建后触发，可在此打开相机。
  - surfaceChanged：在绘图表面变更后触发。
  - surfaceDestroyed：在绘图表面销毁后触发。
- removeCallback：移除绘图表面的回调接口。
- isCreating：判断绘图表面是否有效。如果在别处操作SurfaceView，就要判断当前绘图表面是否有效。
- getSurface：获取绘图表面的对象，即预览界面。
- setFixedSize：设置预览界面的尺寸。
- setFormat：设置绘图表面的格式。

绘图格式的取值说明见表 9-1。

表9-1　绘图格式的取值说明

PixelFormat 类的绘图格式类型	说明
TRANSPAREN	透明
TRANSLUCENT	半透明
OPAQUE	不透明

下面用一个具体的例子说明普通视图与表面视图的区别。图 9-1 和图 9-2 所示为普通视图在 UI 线程中转动扇形区域的效果图，前后两个界面的扇形大小相同，只是角度不同。



图 9-1　普通视图的转动界面 1

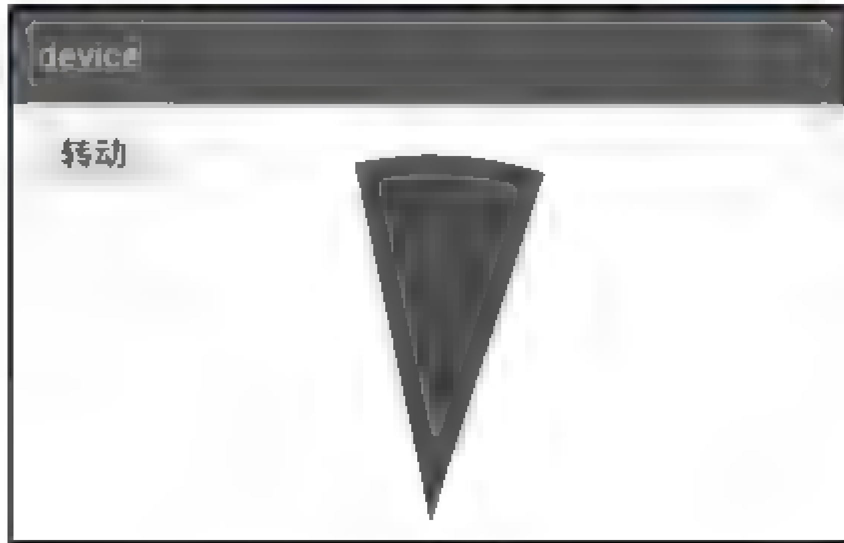


图 9-2　普通视图转的动界面 2

再来看表面视图转动扇形区域的效果图，此时开启了两个线程，一个线程绘制红色扇形，另一个线程绘制青色扇形，前后两个时间点的画面如图 9-3 和图 9-4 所示。

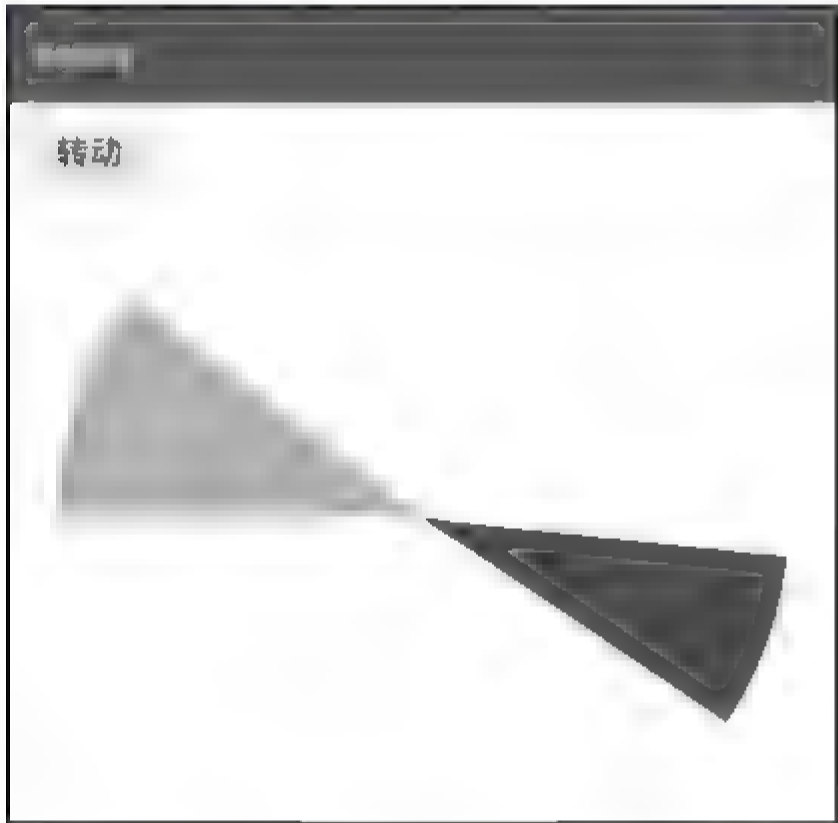


图 9-3　表面视图的转动界面 1

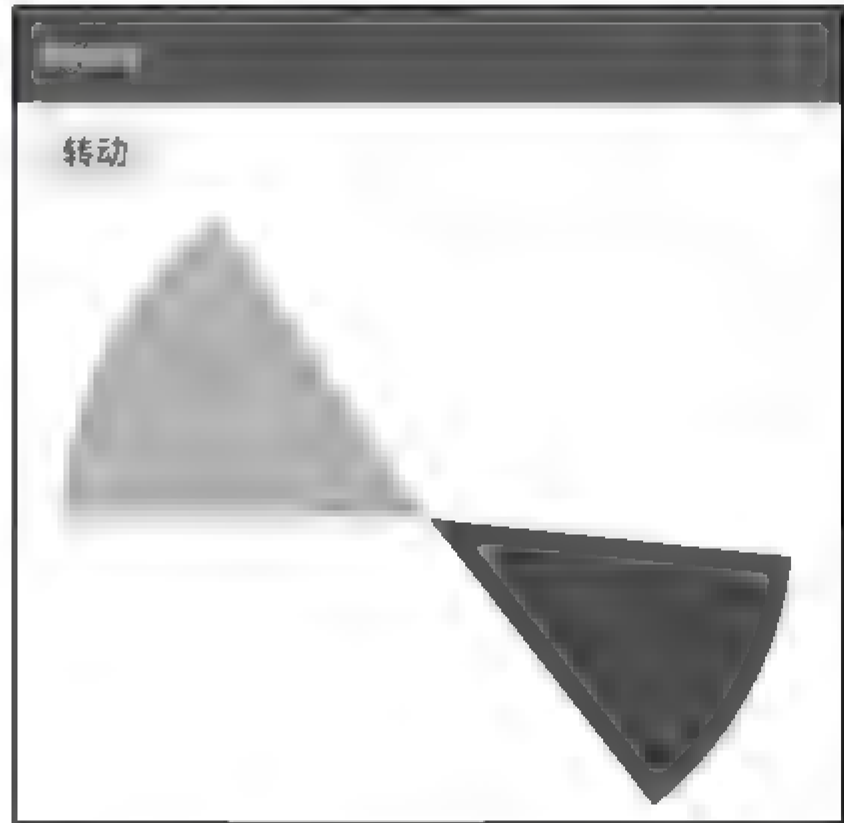


图 9-4　表面视图的转动界面 2

从表面视图的转动效果可以看到，它与普通视图在处理上的区别主要有以下两点：

（1）表面视图允许开启多个线程同时进行绘图操作，而普通视图只有一个 UI 线程可以绘图。

（2）表面视图不会自动清空上次的绘图结果，即绘图操作是增量进行的，而普通视图在每次绘图前都会清空上次的绘图结果。



9.1.2 使用 Camera 拍照

相机 Camera 是直接操作摄像头硬件的工具类，包括后置摄像头和前置摄像头，有以下常用方法。

- getNumberOfCameras：获取本设备的摄像头数目。
- open：打开摄像头，默认打开后置摄像头。如果有多个摄像头，那么 open(0)表示打开后置摄像头，open(1)表示打开前置摄像头。
- getParameters：获取摄像头的拍照参数，返回 Camera.Parameters 对象。
- setParameters：设置摄像头的拍照参数。具体的拍照参数通过调用 Camera.Parameters 的下列方法进行设置。
    - setPreviewSize：设置预览界面的尺寸。
    - setPictureSize：设置保存图片的尺寸。
    - setPictureFormat：设置图片格式。一般使用 ImageFormat.JPEG 表示 JPG 格式。
    - setFocusMode：设置对焦模式。一般使用 Camera.Parameters.FOCUS_MODE_AUTO 表示自动对焦。
- setPreviewDisplay：设置预览界面的表面持有者，即 SurfaceHolder 对象。该方法必须在 SurfaceHolder.Callback 的 surfaceCreated 方法中调用。
- startPreview：开始预览。该方法必须在 setPreviewDisplay 方法之后调用。
- unlock：录像时需要对摄像头解锁，这样摄像头才能持续录像。该方法必须在 startPreview 方法之后调用。
- setDisplayOrientation：设置预览的角度。Android 的 0 度在三点钟的水平位置，而手机屏幕是垂直位置，从水平位置到垂直位置需要旋转 90 度。
- autoFocus：设置对焦事件。参数自动对焦接口 AutoFocusCallback 的 onAutoFocus 方法在对焦完成时触发，在此提示用户对焦完毕可以拍照了。
- takePicture：开始拍照，并设置拍照相关事件。第一个参数为快门回调接口 ShutterCallback，它的 onShutter 方法在按下快门时触发，通常可在此播放拍照声音，默认为“咔嚓”一声；第二个参数的 PictureCallback 表示原始图像的回调接口，通常无须处理直接传 null；第三个参数的 PictureCallback 表示 JPG 图像的回调接口，压缩后的图像数据可在该接口中的 onPictureTaken 方法中获得。
- setZoomChangeListener：设置缩放比例变化事件。缩放变化监听器 OnZoomChangeListener 的 onZoomChange 方法在缩放比例发生变化时触发。
- setPreviewCallback：设置预览回调事件，通常在连拍时调用。预览回调接口 PreviewCallback 的 onPreviewFrame 方法在预览图像发生变化时触发。
- stopPreview：停止预览。
- lock：录像完毕对摄像头加锁。该方法在 stopPreview 方法之后调用。
- release：释放摄像头。因为摄像头不能重复打开，所以每次退出拍照时都要释放摄像头。

结合使用相机工具与表面视图可以实现单拍（每次只拍一张照片）与连拍（自动连续拍

摄多张照片）两种拍照功能。其中，单拍功能的实现代码如下：

```
    public CameraView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        mHolder = getHolder();
        mHolder.setFormat(PixelFormat.TRANSPARENT);
        mHolder.addCallback(mSurfaceCallback);
    }

    private String mPhotoPath;

    //外部调用该方法获得拍摄照片的路径
    public String getPhotoPath() {
        return mPhotoPath;
    }

    //外部调用该方法进行拍照动作
    public void doTakePicture() {
        if(isPreviewing && mCamera!=null) {
            mCamera.takePicture(mShutterCallback, null, mPictureCallback);
        }
    }

    //快门按下的回调，在这里可以设置类似播放“咔嚓”声之类的操作。默认的是“咔嚓”
    private ShutterCallback mShutterCallback = new ShutterCallback() {
        public void onShutter() {
            Log.d(TAG, "onShutter...");
        }
    };

    //获得拍照图片的回调。在此保存图片
    private PictureCallback mPictureCallback = new PictureCallback() {
        public void onPictureTaken(byte[] data, Camera camera) {
            Log.d(TAG, "onPictureTaken...");
            Bitmap raw = null;
            if(null != data) {
                raw = BitmapFactory.decodeByteArray(data, 0, data.length);//将 data 解析成位图
                mCamera.stopPreview();
                isPreviewing = false;
            }
            Bitmap bitmap = BitmapUtil.getRotateBitmap(raw, (mCameraType==
CAMERA_BEHIND)?90:-90);
```



```
                mPhotoPath = String.format("%s%s.jpg", BitmapUtil.getCachePath(mContext),
Utils.getNowDateTime());
                BitmapUtil.saveBitmap(mPhotoPath, bitmap, "jpg", 80);
                Log.d(TAG, "bitmap.size="+(bitmap.getByteCount()/1024)+"K"+", path="+mPhotoPath);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                //再次进入预览
                mCamera.startPreview();
                isPreviewing = true;
            }
        };

        //预览界面状态变更时的回调
        private SurfaceHolder.Callback mSurfaceCallback = new SurfaceHolder.Callback() {
            @Override
            public void surfaceCreated(SurfaceHolder holder) {
                // 当预览视图创建的时候开启相机
                mCamera = Camera.open(mCameraType);
                try {
                    // 设置预览
                    mCamera.setPreviewDisplay(holder);
                    mCameraSize = MetricsUtil.getCameraSize(mCamera.getParameters(),
MetricsUtil.getSize(mContext));
                    Log.d(TAG, "width="+mCameraSize.x+", height="+mCameraSize.y);
                    Camera.Parameters parameters = mCamera.getParameters();
                    // 设置预览大小
                    parameters.setPreviewSize(mCameraSize.x, mCameraSize.y);
                    // 设置图片保存时的分辨率大小
                    parameters.setPictureSize(mCameraSize.x, mCameraSize.y);
                    // 设置格式
                    parameters.setPictureFormat(ImageFormat.JPEG);
                    // 设置自动对焦。前置摄像头似乎无法自动对焦
                    if (mCameraType == CameraView.CAMERA_BEHIND) {
                        parameters.setFocusMode(Camera.Parameters.FOCUS_MODE_AUTO);
                    }
                    mCamera.setParameters(parameters);
                } catch (Exception e) {
                    e.printStackTrace();
                    mCamera.release();
                    mCamera = null;
```

```
            }
            return;
        }

        @Override
        public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
            mCamera.setDisplayOrientation(90);
            mCamera.startPreview();
            isPreviewing = true;
            mCamera.autoFocus(null);
            //setPreviewCallback 给连拍使用
            mCamera.setPreviewCallback(mPreviewCallback);
        }

        @Override
        public void surfaceDestroyed(SurfaceHolder holder) {
            mCamera.setPreviewCallback(null);
            mCamera.stopPreview();
            mCamera.release();
            mCamera = null;
        }
    };
```

单拍的效果如图 9-5 所示，每次从拍照页面返回时都展示最后一张拍摄的照片。

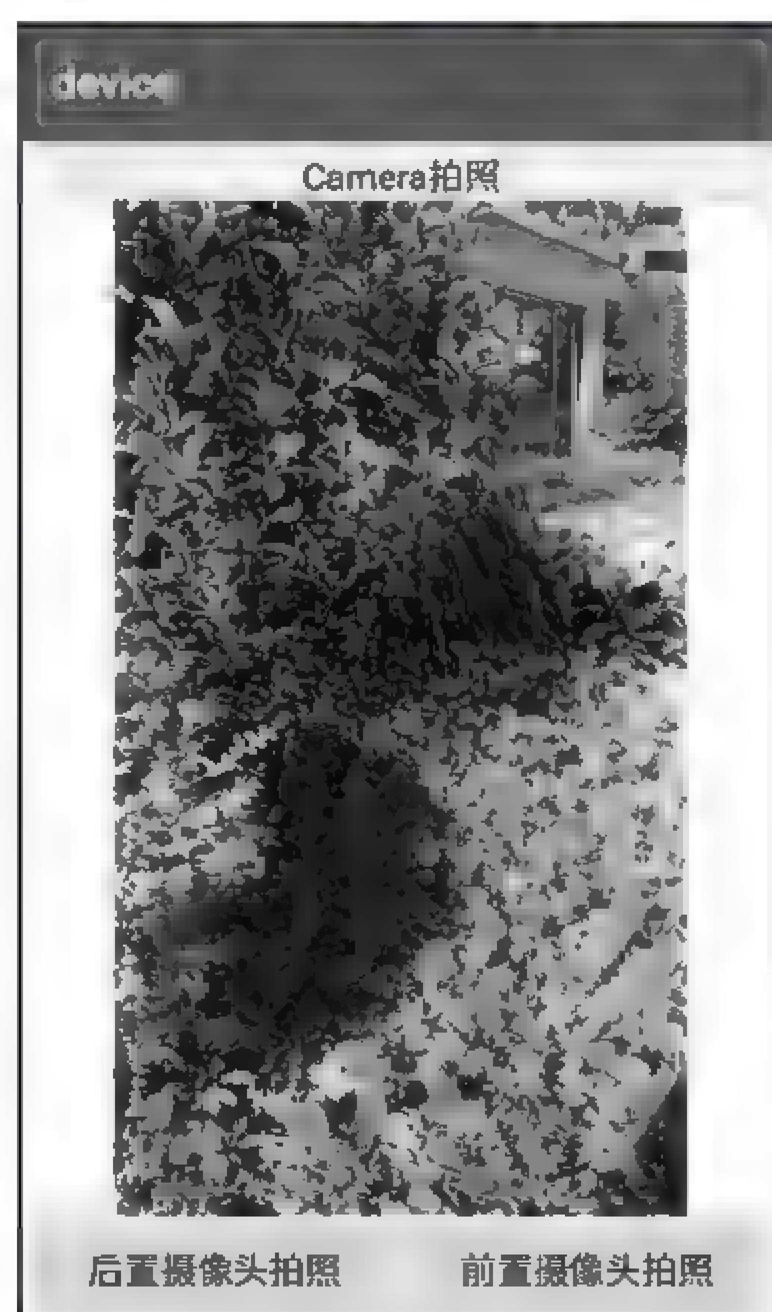


图 9-5　使用 Camera 单拍的效果图

实现连拍功能要先调用 setPreviewCallback 方法设置预览回调接口，然后实现回调接口中的 onPreviewFrame 方法，在该方法中获得并保存每张预览照片。连拍功能的实现代码如下：

```
	private boolean bShooting = false;
	private int shooting_num = 0;
	private ArrayList<String> mShootingArray;

	//外部调用该方法获得连拍照片的路径
	public ArrayList<String> getShootingList() {
		Log.d(TAG, "mShootingArray.size()="+mShootingArray.size());
		return mShootingArray;
	}

	//外部调用该方法进行连拍动作
	public void doTakeShooting() {
		mShootingArray = new ArrayList<String>();
		bShooting = true;
		shooting_num = 0;
	}

	private PreviewCallback mPreviewCallback = new PreviewCallback() {
		@Override
		public void onPreviewFrame(byte[] data, Camera camera) {
			Log.d(TAG, "onPreviewFrame bShooting="+bShooting+", shooting_num="
+shooting_num);
			if (!bShooting) {
				return;
			}
			Camera.Parameters parameters = camera.getParameters();
			int imageFormat = parameters.getPreviewFormat();
			int w = parameters.getPreviewSize().width;
			int h = parameters.getPreviewSize().height;
			Rect rect = new Rect(0, 0, w, h);
			YuvImage yuvImg = new YuvImage(data, imageFormat, w, h, null);
			try {
				ByteArrayOutputStream bos = new ByteArrayOutputStream();
				yuvImg.compressToJpeg(rect, 80, bos);
				Bitmap raw = BitmapFactory.decodeByteArray(bos.toByteArray(), 0, bos.size());
				Bitmap bitmap = BitmapUtil.getRotateBitmap(raw, (mCameraType==
CAMERA_BEHIND)?90:-90);
				String path = String.format("%s%s.jpg", BitmapUtil.getCachePath(mContext),
Utils.getNowDateTimeFull());
				BitmapUtil.saveBitmap(path, bitmap, "jpg", 80);
```

```
                    Log.d(TAG, "bitmap.size="+(bitmap.getByteCount()/1024)+"K"+", path="+path);
                    //再次进入预览
                    camera.startPreview();
                    shooting_num++;
                    mShootingArray.add(path);
                    if (shooting_num > 8) {   //每次连拍 9 张
                        bShooting = false;
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
```

连拍的效果如图 9-6 所示，每次从拍照页面返回时都展示最后一组连拍的照片合集。

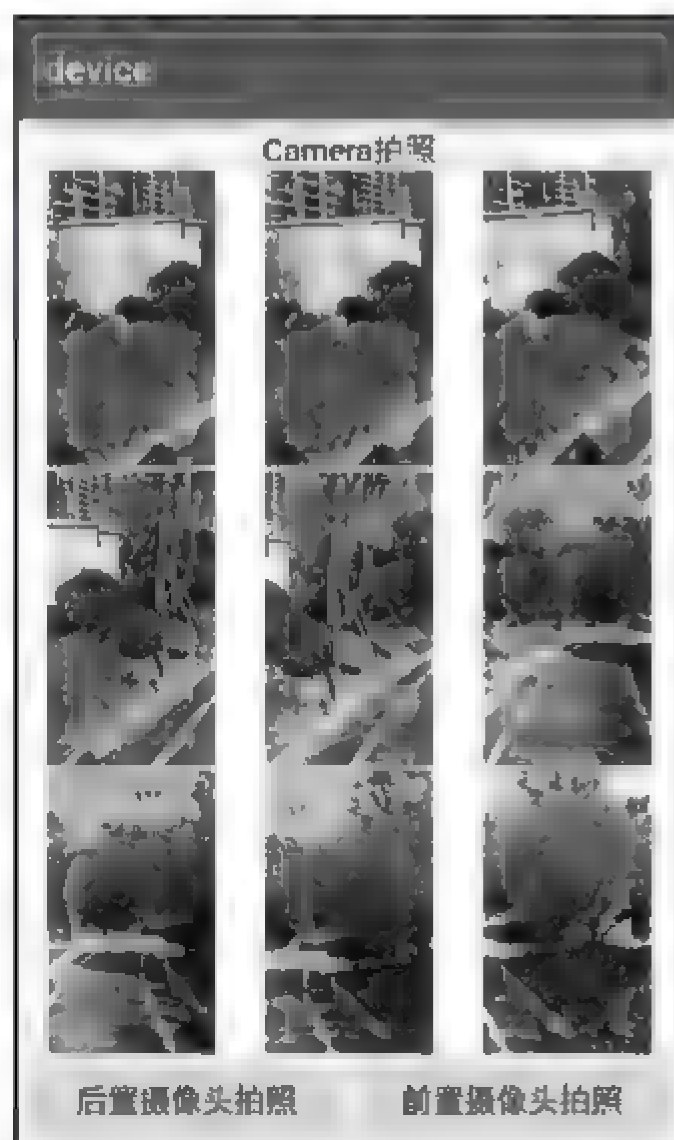


图 9-6　使用 Camera 连拍的效果图

9.1.3　纹理视图 TextureView

表面视图 SurfaceView 在一般情况下足够使用了，但是有一些限制。因为表面视图不是通过 onDraw 方法和 dispatchDraw 方法进行绘图，所以无法使用 View 的基本视图方法。例如，各种视图变化方法均无法奏效，包括透明度变化方法 setAlpha、平移方法 setTranslation、缩放方法 setScale、旋转方法 setRotation 等，甚至连最基础的背景图设置方法 setBackground 都失效了。

为了解决表面视图的不足之处，Android 在 4.0 之后引入了纹理视图 TextureView。与表面视图相比，纹理视图并没有创建一个单独的绘图表面用来绘制，可以像普通视图一样执行变换操作，也可以正常设置背景图。

下面是 TextureView 的常用方法。

- lockCanvas：锁定并获取画布。
- unlockCanvasAndPost：解锁并刷新画布。
- setSurfaceTextureListener：设置表面纹理的监听器。该方法相当于 SurfaceHolder 的 addCallback 方法，用来监控表面纹理的状态变化事件。方法参数为 SurfaceTextureListener 监听器对象，需重写以下4个方法。
  - onSurfaceTextureAvailable：在表面纹理可用时触发，可在此进行打开相机等操作。
  - onSurfaceTextureSizeChanged：在表面纹理尺寸变化时触发。
  - onSurfaceTextureDestroyed：在表面纹理销毁时触发。
  - onSurfaceTextureUpdated：在表面纹理更新时触发。
- isAvailable：判断表面纹理是否可用。
- getSurfaceTexture：获取表面纹理。

下面通过具体例子说明纹理视图与表面视图的区别。图9-7所示为纹理视图的透明度值为0.2，扇形看起来颜色较浅；图9-8所示为纹理视图的透明值增大为0.8，此时扇形的颜色较深。



图9-7　透明度为0.2的纹理视图

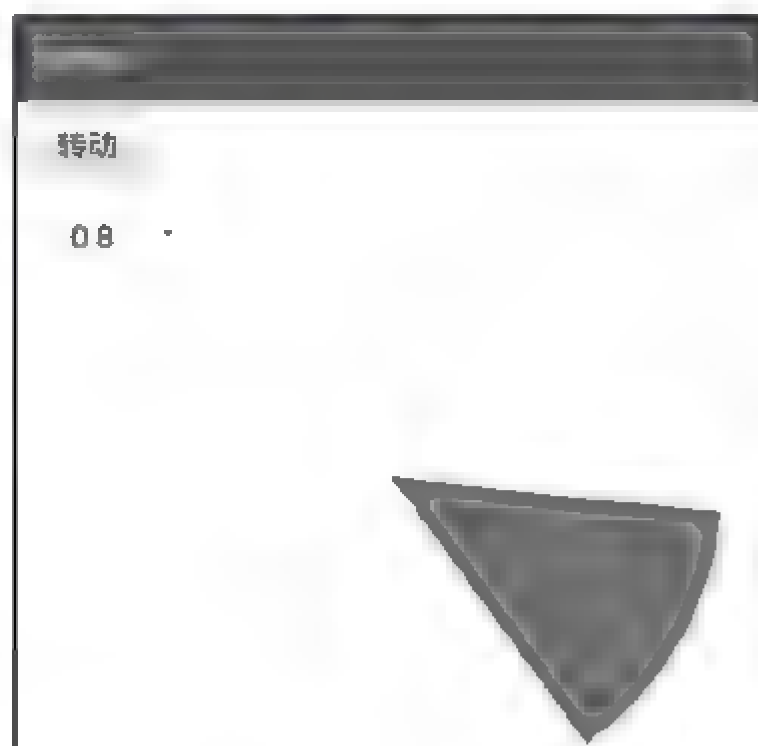


图9-8　透明度为0.8的纹理视图

纹理视图和表面视图的默认背景都是黑色，要想把背景改为白色，TextureView 可以直接调用背景设置方法 setBackground，而 SurfaceView 要调用以下代码才能把背景洗白：

```
setZOrderOnTop(true);
mHolder.setFormat(PixelFormat.TRANSLUCENT);
```

9.1.4　使用 Camera 2 拍照

如同纹理视图是表面视图的升级版那样，Android 在5.0之后推出了 Camera 的升级版——camera 2。按照 Android 的官方说明，camera 2 支持以下5点新特性：

（1）支持每秒30帧的全高清连拍。

（2）支持在每帧之间使用不同的设置。

（3）支持原生格式的图像输出。



（4）支持零延迟快门和电影速拍。

（5）支持相机在其他方面的手动控制，比如设置噪音消除的级别。

camera2 在架构上做了大幅改造，原先的 Camera 类被拆分为多个管理类，主要有相机管理器 CameraManager、相机设备 CameraDevice、相机拍照会话 CameraCaptureSession、图像读取器 ImageReader。

1. 相机管理器 CameraManager

相机管理器用于获取可用摄像头列表、打开摄像头等，对象从系统服务 CAMERA_SERVICE 获取。常用方法说明如下：

- getCameraIdList：获取相机列表。通常返回两条记录，一条是后置摄像头，另一条是前置摄像头。
- getCameraCharacteristics：获取相机的参数信息。包括相机的支持级别、照片的尺寸等。

因为 camera2 是 Android5.0 之后才有的新特性，不少手机还不能很好的支持，所以最好先检查相机的支持级别，如果返回值为 INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY，就不建议在 App 中使用 camera2 的相关技术。检查相机支持级别的代码如下：

```
CameraCharacteristics cc = cm.getCameraCharacteristics(cameraid);
// CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_FULL 表示完全支持
// CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LIMITED 表示有限支持
// CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY 表示遗留的
int level = cc.get(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL);
```

- openCamera：打开指定摄像头，第一个参数为指定摄像头的 id，第二个参数为设备状态监听器，该监听器需实现接口 CameraDevice.StateCallback 的 onOpened 方法（方法内部再调用 CameraDevice 对象的 createCaptureRequest 方法）。
- setTorchMode：在不打开摄像头的情况下，开启或关闭闪光灯。为 true 表示开启闪光灯，为 false 表示关闭闪光灯。

2. 相机设备 CameraDevice

相机设备用于创建拍照请求、添加预览界面、创建拍照会话等。常用方法说明如下：

- createCaptureRequest：创建拍照请求，第二个参数为会话状态的监听器，该监听器需实现会话状态回调接口 CameraCaptureSession.StateCallback 的 onConfigured 方法（方法内部再调用 CameraCaptureSession 对象的 setRepeatingRequest 方法，将预览影像输出到屏幕）。createCaptureRequest 方法返回一个 CaptureRequest 的预览对象。
- close：关闭相机。

3. 相机拍照会话 CameraCaptureSession

相机拍照会话用于设置单拍会话（每次只拍一张照片）、连拍会话（自动连续拍摄多张照片）等。常用方法说明如下：

- getDevice：获得该会话的相机设备对象。
- capture：拍照并输出到指定目标。输出目标为 CaptureRequest 对象时，表示显示在屏幕上；输出目标为 ImageReader 对象时，表示要保存照片。
- setRepeatingRequest：设置连拍请求并输出到指定目标。输出目标为 CaptureRequest 对象时，表示显示在屏幕上；输出目标为 ImageReader 对象时，表示要保存照片。
- stopRepeating：停止连拍。

4. 图像读取器 ImageReader

图像读取器用于获取并保存照片信息，一旦有图像数据生成，立刻触发 onImageAvailable 方法。常用方法说明如下：

- getSurface：获得图像读取的表面对象。
- setOnImageAvailableListener：设置图像数据的可用监听器。该监听器需实现接口 ImageReader.OnImageAvailableListener 的 onImageAvailable 方法。

这几个相机类之间的调用流程比原来的 Camera 类要复杂许多，详细的文字说明反而不容易理解，限于篇幅这里就不贴出大段的调用代码了，读者可翻阅本书下载资源里的 camera 2 调用模板，一边阅读代码、一边熟悉调用流程。

使用 camera 2 拍照的效果如图 9-9 和图 9-10 所示。其中，图 9-9 所示为单拍时拍摄的最后一张照片，图 9-10 所示为连拍时拍摄的最后一组照片合集。



图 9-9 使用 Camera2 单拍的效果图



图 9-10 使用 Camera2 连拍的效果图

9.2 麦 克 风

本节介绍以麦克风为基础的声效应用，首先简要说明拖动条 SeekBar 的用法，描述如何使

用拖动条调整各类音量大小；然后介绍媒体录制器 MediaRecorder 与媒体播放器 MediaPlayer，并演示通过媒体录制器和媒体播放器完成录音和播音功能；最后结合 9.1 节的相机与表面视图知识演示通过媒体录制器和媒体播放器完成录像和放映功能。

9.2.1 拖动条 SeekBar

拖动条 SeekBar 继承自进度条 ProcessBar，与进度条的不同之处在于：进度条只能在代码中修改进度，不能由用户改变进度值；拖动条不但可以在代码中修改进度，还可以由用户在屏幕上通过拖动操作改变进度。拖动条可用于音频和视频播放时的进度条，用户通过拖动操作控制播放器快进或快退到指定位置，然后从新位置开始播放音频或视频。除此之外，拖动条还可调节各种音量大小、调节屏幕亮度、调节字体大小等。

下面是 SeekBar 新增加的 4 个方法。

- setThumb：设置当前进度位置的图标。
- setThumbOffset：设置当前进度图标的偏移量。
- setKeyProgressIncrement：设置使用方向键更改进度时每次的增加值。
- setOnSeekBarChangeListener：设置拖动变化事件。需实现监听器 OnSeekBarChangeListener 的 3 个方法。
    - onProgressChanged：在进度变化时触发。第 3 个参数表示是否来自用户，为 true 表示用户拖动，为 false 表示代码修改进度。
    - onStartTrackingTouch：开始拖动时触发。
    - onStopTrackingTouch：结束拖动时触发。一般在该方法中添加用户拖动的处理逻辑。

下面是操作拖动条的代码：

```
public class SeekbarActivity extends AppCompatActivity implements OnSeekBarChangeListener {
	private SeekBar sb_progress;
	private TextView tv_progress;

	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_seekbar);
		tv_progress = (TextView) findViewById(R.id.tv_progress);
		sb_progress = (SeekBar) findViewById(R.id.sb_progress);
		sb_progress.setOnSeekBarChangeListener(this);
		sb_progress.setProgress(50);
	}

	@Override
	public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
		String desc = "当前进度为 ："+seekBar.getProgress()+"，最大进度为"+seekBar.getMax();
		tv_progress.setText(desc);
```

```
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
        }
    }
```

上述代码的界面效果如图 9-11 和图 9-12 所示。其中，图 9-11 所示为拖动前的界面，进度值为 50；图 9-12 所示为向右拖动后的界面，进度值为 73。



图 9-11　拖动前的 SeekBar



图 9-12　拖动后的 SeekBar

9.2.2　音量控制

Android 只有一个麦克风，却有 6 类铃音，分别是通话音、系统音、铃音、媒体音、闹钟音、通知音，铃音类型的取值说明见表 9-2。

表9-2　铃音类型的取值说明

AudioManager 类的铃音类型	铃音名称	说明
STREAM_VOICE_CALL	通话音	
STREAM_SYSTEM	系统音	
STREAM_RING	铃音	来电与收短信的铃声
STREAM_MUSIC	媒体音	音频、视频、游戏等的声音
STREAM_ALARM	闹钟音	
STREAM_NOTIFICATION	通知音	

管理这些铃声音量的工具是 AudioManager，对象从系统服务 AUDIO_SERVICE 中获取。下面是 AudioManager 的常用方法。

- getStreamMaxVolume：获取指定类型铃声的最大音量。
- getStreamVolume：获取指定类型铃声的当前音量。
- getRingerMode：获取指定类型铃声的响铃模式。响铃模式的取值说明见表 9-3。

表9-3 响铃模式的取值说明

AudioManager 类的响铃模式	说明
RINGER_MODE_NORMAL	正常
RINGER_MODE_SILENT	静音
RINGER_MODE_VIBRATE	震动

- setStreamVolume：设置指定类型铃声的当前音量。
- setRingerMode：设置指定类型铃声的响铃模式。响铃模式的取值说明见表 9-3。
- adjustStreamVolume：调整指定类型铃声的当前音量。第一个参数是铃声类型；第二个参数是调整方向，音量调整方向的取值说明见表 9-4；第三个参数表示调整时的附加动作，一般使用 FLAG_PLAY_SOUND 表示调整时提示一个铃声。

表9-4 音量调整方向的取值说明

AudioManager 类的音量调整方向	说明
ADJUST_RAISE	调大一级
ADJUST_LOWER	调小一级
ADJUST_SAME	保持不变
ADJUST_MUTE	静音
ADJUST_UNMUTE	取消静音
ADJUST_TOGGLE_MUTE	静音取反，即原来不是静音就设置静音，原来是静音就取消静音

上面的 setStreamVolume 和 adjustStreamVolume 两个方法都能用来设置音量，不同的是 setStreamVolume 直接将音量调整到目标值，通常与拖动条配合使用；而 adjustStreamVolume 是以当前音量为基础，然后调大、调小或调静音。

音量调整的效果如图 9-13 所示，这个设置页面不但允许直接调整音量到目标值，还允许逐级调大或逐级调小音量。

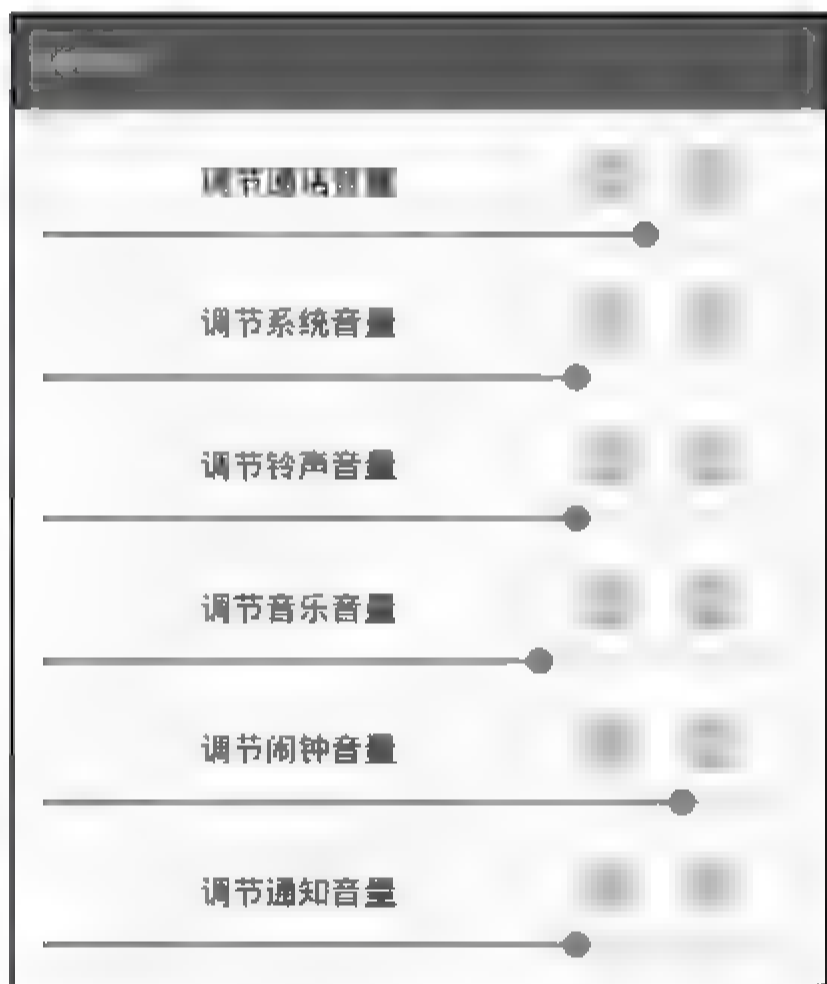


图 9-13 各种铃音的音量调整界面

9.2.3 录音与播音

Android 中没有单独操作麦克风的工具类，如果要录音就用媒体录制器 MediaRecorder，如果要播音就用媒体播放器 MediaPlayer 类。下面分别进行介绍。

1. 媒体录制器 MediaRecorder

MediaRecorder 是 Android 自带的音频和视频录制工具，它通过操纵摄像头和麦克风完成媒体录制，既可录制视频，又可单独录制音频。

下面是 MediaRecorder 的常用方法（录音与录像通用）。

- reset：重置录制资源。
- prepare：准备录制。
- start：开始录制。
- stop：结束录制。
- release：释放录制资源。
- setOnErrorListener：设置错误监听器。可监听服务器异常和未知错误的事件。需要实现接口 MediaRecorder.OnErrorListener 的 onError 方法。
- setOnInfoListener：设置信息监听器。可监听录制结束事件，包括达到录制时长或达到录制大小。需要实现接口 MediaRecorder.OnInfoListener 的 onInfo 方法。
- setMaxDuration：设置可录制的最大时长，单位毫秒。
- setMaxFileSize：设置可录制的最大文件大小，单位字节。
- setOutputFile：设置输出文件的路径。

下面是 MediaRecorder 用于音频录制的方法（当然录像时要一起录音）。

- setAudioSource：设置音频来源。一般使用麦克风 AudioSource.MIC。
- setOutputFormat：设置媒体输出格式。媒体输出格式的取值说明见表 9-5。

表9-5 媒体输出格式的取值说明

OutputFormat 类的输出格式	格式分类	扩展名	格式说明
AMR_NB	音频	.amr	窄带格式
AMR_WB	音频	.amr	宽带格式
AAC_ADTS	音频	.aac	高级的音频传输流格式
MPEG_4	视频	.mp4	MPEG4 格式
THREE_GPP	视频	.3gp	3GP 格式

- setAudioEncoder：设置音频编码器。音频编码器的取值说明见表 9-6。注意：该方法应在 setOutputFormat 方法之后执行，否则会出现 setAudioEncoder called in an invalid state(2)的异常。

表9-6 音频编码器的取值说明

AudioEncoder 类的音频编码器	说明
AMR_NB	窄带编码
AMR_WB	宽带编码
AAC	低复杂度的高级编码
HE_AAC	高效率的高级编码
AAC_ELD	高效率的高级编码

- setAudioSamplingRate：设置音频的采样率，单位千赫兹（kHz）。AMR_NB 格式默认 8kHz，AMR_WB 格式默认 16kHz。
- setAudioChannels：设置音频的声道数。1 表示单声道，2 表示双声道。
- setAudioEncodingBitRate：设置音频每秒录制的字节数。数值越大音频越清晰。

下面是使用 MediaRecorder 实现简单音频录制器的代码：

```
public class AudioRecorder extends LinearLayout implements OnErrorListener,
        OnInfoListener, OnCheckedChangeListener {
    private MediaRecorder mMediaRecorder;
    private ProgressBar pb_record;
    private CheckBox ck_record;
    private Timer mTimer; // 计时器
    private int mRecordMaxTime = 10; // 一次录音最长时间
    private int mTimeCount; // 时间计数
    private String mRecordFilePath;

    public AudioRecorder(Context context) {
        this(context, null);
    }

    public AudioRecorder(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public AudioRecorder(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        LayoutInflater.from(context).inflate(R.layout.audio_recorder, this);
        pb_record = (ProgressBar) findViewById(R.id.pb_record);
        pb_record.setMax(mRecordMaxTime);
        ck_record = (CheckBox) findViewById(R.id.ck_record);
        ck_record.setOnCheckedChangeListener(this);
    }

    // 开始录音
    public void start() {
        mRecordFilePath = MediaUtil.getRecordFilePath("RecordAudio", ".amr");
        try {
            initRecord();
            mTimeCount = 0;
            mTimer = new Timer();
            mTimer.schedule(new TimerTask() {
                @Override
                public void run() {
                    pb_record.setProgress(mTimeCount++);
                }
            }, 0, 1000);
        } catch (Exception e) {
```



```
            e.printStackTrace();
        }
    }

    // 停止录音
    public void stop() {
        if (mOnRecordFinishListener != null) {
            mOnRecordFinishListener.onRecordFinish();
        }
        stopRecord();
        releaseRecord();
    }

    // 获得最近一次的录音文件路径
    public String getRecordFilePath() {
        return mRecordFilePath;
    }

    private void initRecord() {
        mMediaRecorder = new MediaRecorder();
        mMediaRecorder.setOnErrorListener(this);
        mMediaRecorder.setOnInfoListener(this);
        mMediaRecorder.setAudioSource(AudioSource.MIC); // 音频源
        mMediaRecorder.setOutputFormat(OutputFormat.AMR_NB);
        mMediaRecorder.setAudioEncoder(AudioEncoder.AMR_NB); // 音频格式
        mMediaRecorder.setMaxDuration(10 * 1000); // 设置录制时长
        mMediaRecorder.setOutputFile(mRecordFilePath);
        try {
            mMediaRecorder.prepare();
            mMediaRecorder.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void stopRecord() {
        pb_record.setProgress(0);
        if (mTimer != null) {
            mTimer.cancel();
        }
        if (mMediaRecorder != null) {
            mMediaRecorder.setOnErrorListener(null);
            mMediaRecorder.setPreviewDisplay(null);
```

```
            try {
                mMediaRecorder.stop();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private void releaseRecord() {
        if (mMediaRecorder != null) {
            mMediaRecorder.setOnErrorListener(null);
            mMediaRecorder.release();
            mMediaRecorder = null;
        }
    }

    private OnRecordFinishListener mOnRecordFinishListener; // 录制完成回调接口
    public interface OnRecordFinishListener {
        public void onRecordFinish();
    }

    public void setOnRecordFinishListener(OnRecordFinishListener listener) {
        mOnRecordFinishListener = listener;
    }

    @Override
    public void onError(MediaRecorder mr, int what, int extra) {
        if (mr != null) {
            mr.reset();
        }
    }

    @Override
    public void onInfo(MediaRecorder mr, int what, int extra) {
        if (what == MediaRecorder.MEDIA_RECORDER_INFO_MAX_DURATION_REACHED
                || what == MediaRecorder.MEDIA_RECORDER_INFO_MAX_FILESIZE_REACHED) {
            ck_record.setChecked(false);
        }
    }

    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (buttonView.getId() == R.id.ck_record) {
```

```
                if (isChecked == true) {
                    ck_record.setText("停止录制");
                    start();
                } else {
                    ck_record.setText("开始录制");
                    stop();
                }
            }
        }
    }
```

另外，注意录音与录像需要在 AndroidManifest.xml 中添加权限（录制操作通常会保存媒体文件，也就是操作 SD 卡，所以需要加上 SD 卡的读写权限）：

```
<!-- 录像/录音 -->
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<!-- SD 卡 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

2. 媒体播放器 MediaPlayer

MediaPlayer 是 Android 自带的音频和视频播放器，可用于播放 MediaRecorder 录制的媒体文件，包括表 9-5 所示的文件格式，以及 MP3、WAV、MID、OGG 等音频文件。

下面是 MediaPlayer 的常用方法（播音与放映通用）。

- reset：重置播放器。
- prepare：准备播放。
- start：开始播放。
- pause：暂停播放。
- stop：停止播放。
- setOnPreparedListener：设置准备播放监听器。需要实现接口 MediaPlayer.OnPreparedListener 的 onPrepared 方法。
- setOnCompletionListener：设置结束播放监听器。需要实现接口 MediaPlayer.OnCompletionListener 的 onCompletion 方法。
- setOnSeekCompleteListener：设置播放拖动监听器。需要实现接口 MediaPlayer.OnSeekCompleteListener 的 onSeekComplete 方法。
- create：创建指定 Uri 的播放器。
- setDataSource：设置播放数据来源的文件路径。create 与 setDataSource 两个方法只需调用一个。

- setVolume：设置音量。两个参数分别是左声道和右声道的音量，取值在 0～1 之间。
- setAudioStreamType：设置音频流的类型。音频流类型的取值说明见表 9-2。
- setLooping：设置是否循环播放。true 表示循环播放，false 表示只播放一次。
- isPlaying：判断是否正在播放。
- seekTo：拖动播放进度到指定位置。该方法可与拖动条 SeekBar 配合使用。
- getCurrentPosition：获取当前播放进度所在的位置。
- getDuration：获取播放时长，单位毫秒。

下面是使用 MediaPlayer 实现简单音频播放器的代码：

```
public class AudioPlayer extends LinearLayout implements OnCompletionListener, OnCheckedChangeListener {
    private static final String TAG = "AudioPlayer";
    private MediaPlayer mMediaPlayer;
    private ProgressBar pb_play;
    private CheckBox ck_play;
    private Timer mTimer; // 计时器
    private String mAudioPath;
    private boolean bFinish = true;

    public AudioPlayer(Context context) {
        this(context, null);
    }

    public AudioPlayer(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public AudioPlayer(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        LayoutInflater.from(context).inflate(R.layout.audio_player, this);
        pb_play = (ProgressBar) findViewById(R.id.pb_play);
        ck_play = (CheckBox) findViewById(R.id.ck_play);
        ck_play.setOnCheckedChangeListener(this);
    }

    // 初始化音频文件路径与播放器
    public void init(String path) {
        mAudioPath = path;
        ck_play.setEnabled(true);
        ck_play.setTextColor(Color.BLACK);
        mMediaPlayer = new MediaPlayer();
        mMediaPlayer.setOnCompletionListener(this);
    }
```

```
private void play() {
    try {
        mMediaPlayer.reset();
        mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
        Log.d(TAG, "audio path = " + mAudioPath);
        // 录制完毕要等一秒钟再调用 setDataSource 方法，因为此时可能尚未完成写入
        mMediaPlayer.setDataSource(mAudioPath);
        mMediaPlayer.prepare();
        mMediaPlayer.start();
        pb_play.setMax(mMediaPlayer.getDuration());
        mTimer = new Timer();
        mTimer.schedule(new TimerTask() {
            @Override
            public void run() {
                pb_play.setProgress(mMediaPlayer.getCurrentPosition());
            }
        }, 0, 1000);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

@Override
public void onCompletion(MediaPlayer mp) {
    bFinish = true;
    ck_play.setChecked(false);
    if (mTimer != null) {
        mTimer.cancel();
    }
}

@Override
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    if (buttonView.getId() == R.id.ck_play) {
        if (isChecked == true) {
            ck_play.setText("暂停播放");
            if (bFinish == true) {
                play();
            } else {
                mMediaPlayer.start();
            }
            bFinish = false;
```

```
                } else {
                    ck_play.setText("开始播放");
                    mMediaPlayer.pause();
                }
            }
        }
    }
```

由于音频本身没有对应的界面，因此只能使用进度条间接表达音频录制与播放进度。录音与播音的效果如图 9-14 和图 9-15 所示。其中，图 9-14 表示当前正在录音，图 9-15 表示当前正在播音。



图 9-14　正在录音的界面



图 9-15 正在播音的界面

9.2.4　录像与放映

Android 录制视频与录音一样都使用媒体录制器 MediaRecorder，播放视频与播音一样都使用媒体播放器 MediaPlayer。MediaRecorder 和 MediaPlayer 处理音频与视频的大部分方法相同，不同的是录像与放映多出了对摄像头、表面视图以及视频进行编码和解码的操作。下面分别介绍 MediaRecorder 和 MediaPlayer 对视频的额外处理部分。

1. 媒体录制器 MediaRecorder（录像部分）

下面是 MediaRecorder 录制视频的专用方法（如果只是录音，就不需要这些方法）。

- setCamera：设置相机对象。
- setPreviewDisplay：设置预览界面。预览界面对象可通过 SurfaceHolder 对象的 getSurface 方法获得。
- setOrientationHint：设置预览的角度。跟拍照一样设置为 90，表示界面从水平方向到垂直方向旋转 90 度。
- setVideoSource：设置视频来源。一般使用 VideoSource.CAMERA 表示摄像头。
- setOutputFormat：设置媒体输出格式。媒体输出格式的取值说明见表 9-5。
- setVideoEncoder：设置视频编码器。一般使用 VideoEncoder.MPEG_4_SP 表示 MPEG4 编码。


注 意

该方法要在 setOutputFormat 方法之后调用，否则会报错 java.lang.IllegalStateException。



- setVideoSize：设置视频的分辨率。
- setVideoFrameRate：设置视频每秒录制的帧数。越大视频越连贯，当然最终生成的视频文件也越大。
- setVideoEncodingBitRate：设置视频每秒录制的字节数。越大视频越清晰，setVideoFrameRate 与 setVideoEncodingBitRate 设置一个即可。

录像与录音相比，在界面上增加了 SurfaceView，代码增加了对 SurfaceHolder 与 Camera 的处理。下面是增加的代码片段：

```
private SurfaceHolder mHolder;
private SurfaceHolder.Callback mSurfaceCallback = new SurfaceHolder.Callback() {
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        initCamera();
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        freeCamera();
    }
};

private void initCamera() {
    if (mCamera != null) {
        freeCamera();
    }
    try {
        mCamera = Camera.open();
        mCamera.setDisplayOrientation(90);
        mCamera.setPreviewDisplay(mHolder);
        mCamera.startPreview();
        mCamera.unlock();
    } catch (Exception e) {
        e.printStackTrace();
        freeCamera();
    }
}

private void freeCamera() {
    if (mCamera != null) {
        mCamera.stopPreview();
```

```
                mCamera.lock();
                mCamera.release();
                mCamera = null;
            }
        }

        private void initRecord() {
            mHolder = sv_record.getHolder();
            mHolder.addCallback(mSurfaceCallback);
        }

        private void startRecord() {
            mMediaRecorder = new MediaRecorder();
            mMediaRecorder.setCamera(mCamera);
            mMediaRecorder.setOnErrorListener(this);
            mMediaRecorder.setOnInfoListener(this);
            mMediaRecorder.setPreviewDisplay(mHolder.getSurface());
            mMediaRecorder.setVideoSource(VideoSource.CAMERA);           // 视频源
            mMediaRecorder.setAudioSource(AudioSource.MIC);              // 音频源
            mMediaRecorder.setOutputFormat(OutputFormat.MPEG_4);         // 视频输出格式
            mMediaRecorder.setAudioEncoder(AudioEncoder.AMR_NB);         // 音频格式
            // 如果录像报错 ：MediaRecorder start failed：-19
            // 可注释 setVideoSize 和 setVideoFrameRate，因为尺寸设置必须为摄像头所支持，否则报错
            mMediaRecorder.setVideoEncodingBitRate(1 * 1024 * 512);      // 设置帧频率
            mMediaRecorder.setOrientationHint(90);                    // 输出旋转 90 度，保持竖屏录制
            mMediaRecorder.setVideoEncoder(VideoEncoder.MPEG_4_SP);      // 视频录制格式
            mMediaRecorder.setMaxDuration(mRecordMaxTime * 1000);        // 设置录制时长
            // setMaxFileSize 与 setMaxDuration 设置一个即可
            mMediaRecorder.setOutputFile(mRecordFilePath);
            try {
                mMediaRecorder.prepare();
                mMediaRecorder.start();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        private void stopRecord() {
            pb_record.setProgress(0);
            if (mTimer != null) {
                mTimer.cancel();
            }
            if (mMediaRecorder != null) {
                mMediaRecorder.setOnErrorListener(null);
```



```
                mMediaRecorder.setPreviewDisplay(null);
                try {
                    mMediaRecorder.stop();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }

        private void releaseRecord() {
            if (mMediaRecorder != null) {
                mMediaRecorder.setOnErrorListener(null);
                mMediaRecorder.release();
                mMediaRecorder = null;
            }
        }
```

2. 媒体播放器 MediaPlayer（放映部分）

下面是 MediaPlayer 播放视频的专用方法（如果只是播音，就不需要这些方法）。

- setDisplay：设置播放界面，参数为 SurfaceHolder 类型。
- setSurface：设置播放表层，参数可通过 SurfaceHolder 对象的 getSurface 方法获得。setDisplay 与 setSurface 两个方法只需调用一个。
- setScreenOnWhilePlaying：设置是否使用 SurfaceHolder 显示，也就是是否保持屏幕高亮，从而持续播放视频。为 true 时只能调用 setDisplay，不能调用 setSurface。
- setVideoScalingMode：设置视频的缩放模式，默认为 MediaPlayer.VIDEO_SCALING_MODE_SCALE_TO_FIT，表示固定宽高。
- setOnVideoSizeChangedListener：设置视频缩放监听器。需要实现接口 MediaPlayer.OnVideoSizeChangedListener 的 onVideoSizeChanged 方法。

放映与播音相比，在界面上增加了 SurfaceView，所以布局文件要增加声明 SurfaceView，代码也增加了对 SurfaceView 的处理。下面是修改 play 方法的代码片段，其他部分代码与音频播放类似。

```
        private void play() {
            try {
                mMediaPlayer.reset();
                mMediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
                Log.d(TAG, "video path = " + mVideoPath);
                // 录制完毕要等一秒钟再 setDataSource，因为有可能此时视频文件尚未完全写入
                // 否则会报异常 java.io.IOException：setDataSourceFD failed
                mMediaPlayer.setDataSource(mVideoPath);
                // 把视频界面输出到 SurfaceView
```

```
                mMediaPlayer.setDisplay(sv_play.getHolder());
                mMediaPlayer.prepare();
                mMediaPlayer.start();
                pb_play.setMax(mMediaPlayer.getDuration());
                mTimer = new Timer();
                mTimer.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        pb_play.setProgress(mMediaPlayer.getCurrentPosition());
                    }
                }, 0, 1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

视频录制与播放的效果如图 9-16 和图 9-17 所示。其中，图 9-16 所示为录像时的界面，图 9-17 所示为放映时的界面。



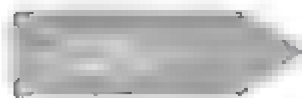
图 9-16　录制视频时的效果图



图 9-17　播放视频时的效果图

9.3 传　感　器

本节介绍常见传感器的用法与相关应用场景，首先列举 Android 目前支持的传感器种类，然后对常用传感器分别进行说明，包括加速度传感器的用法和摇一摇的实现、磁场传感器的用法和指南针的实现、计步传感器的用法和计步器的实现、光线传感器的用法和感光器的实现等。



9.3.1 传感器的种类

传感器 Sensor 是一系列感应器的总称，是 Android 设备用来感知周围环境和运动信息的工具。因为具体的感应信息依赖于相关硬件，所以虽然 Android 定义了众多感应器，但是不是每部手机都能支持这么多感应器，千元以下的低端手机往往只支持加速度等少数感应器。

传感器一般借助于硬件监听环境信息改变，有时会结合软件监听用户的运动信息。目前，Android 支持的传感器类型见表 9-7。

表9-7 传感器类型的取值说明

编号	Sensor 类的传感器类型	传感器名称	说明
1	TYPE_ACCELEROMETER	加速度	常用于摇一摇功能
2	TYPE_MAGNETIC_FIELD	磁场	
3	TYPE_ORIENTATION	方向	已弃用，取而代之的是 getOrientation 方法
4	TYPE_GYROSCOPE	陀螺仪	用来感应手机的旋转和倾斜
5	TYPE_LIGHT	光线	用来感应手机正面的光线强弱
6	TYPE_PRESSURE	压力	用来感应气压
7	TYPE_TEMPERATURE	温度	已弃用，取而代之的是类型 13
8	TYPE_PROXIMITY	距离	
9	TYPE_GRAVITY	重力	
10	TYPE_LINEAR_ACCELERATION	线性加速度	
11	TYPE_ROTATION_VECTOR	旋转矢量	
12	TYPE_RELATIVE_HUMIDITY	相对湿度	
13	TYPE_AMBIENT_TEMPERATURE	环境温度	
14	TYPE_MAGNETIC_FIELD_UNCALIBRATED	无标定磁场	
15	TYPE_GAME_ROTATION_VECTOR	无标定旋转矢量	
16	TYPE_GYROSCOPE_UNCALIBRATED	未校准陀螺仪	
17	TYPE_SIGNIFICANT_MOTION	特殊动作	
18	TYPE_STEP_DETECTOR	步行检测	用户每走一步就触发一次事件
19	TYPE_STEP_COUNTER	步行计数	记录激活后的步伐数
20	TYPE_GEOMAGNETIC_ROTATION_VECTOR	地磁旋转矢量	
21	TYPE_HEART_RATE	心跳速率	可穿戴设备使用，如手环
22	TYPE_TILT_DETECTOR	倾斜检测	
23	TYPE_WAKE_GESTURE	唤醒手势	
24	TYPE_GLANCE_GESTURE	掠过手势	
25	TYPE_PICK_UP_GESTURE	拾起手势	

查看当前设备支持的传感器种类，可通过调用 SensorManager 对象的 getSensorList 方法获得，具体代码如下：

```
private void showSensorInfo() {
    mSensroMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    List<Sensor> sensorList = mSensroMgr.getSensorList(Sensor.TYPE_ALL);
    String show_content = "当前支持的传感器包括 ：\n";
    for (Sensor sensor ： sensorList) {
        if (sensor.getType() >= mSensorType.length) {
            continue;
        }
        mapSensor.put(sensor.getType(), sensor.getName());
    }
    for (Map.Entry<Integer, String> item_map ： mapSensor.entrySet()) {
        int type = item_map.getKey();
        String name = item_map.getValue();
        String content = String.format("%d %s ： %s\n", type, mSensorType[type-1], name);
        show_content += content;
    }
    tv_sensor.setText(show_content);
}
```

图 9-18 所示为某品牌手机上支持的传感器列表，包含目前 Android 系统定义的大部分传感器。



图 9-18　某品牌手机上支持的传感器列表

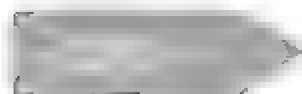
9.3.2　加速度传感器

加速度传感器是最常见的感应器，大部分智能手机都内置了加速度传感器。加速度传感器运用最广泛的功能是微信的摇一摇，用户通过摇晃手机寻找周围的人，其他类似的应用还摇骰子、玩游戏等。

下面以摇一摇的实现演示传感器开发的步骤。

步骤 01 声明一个 SensorManager 对象，该对象从系统服务 SENSOR_SERVICE 中获取实例。

步骤 02 重写 Activity 的 onResume 方法，在该方法中注册传感器监听事件，并指定待监听的传感器类型。例如，摇一摇功能要注册加速度传感器，代码示例如下：



```
mSensroMgr.registerListener(this,
        mSensroMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
        SensorManager.SENSOR_DELAY_NORMAL);
```

步骤 03 重写 Activity 的 onPause 方法，在该方法中注销传感器事件，代码示例如下：

```
mSensroMgr.unregisterListener(this);
```

步骤 04 编写一个传感器事件监听器，该监听器继承自 SensorEventListener，同时需实现 onSensorChanged 和 onAccuracyChanged 两个方法。其中，前一个方法在感应信息变化时触发，业务逻辑都在这里处理；后一个方法在精度改变时触发，一般无须处理。

下面是使用加速度传感器实现简单摇一摇的完整代码：

```
public class AccelerationActivity extends AppCompatActivity implements SensorEventListener {
    private TextView tv_shake;
    private SensorManager mSensroMgr;
    private Vibrator mVibrator;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_acceleration);
        tv_shake = (TextView) findViewById(R.id.tv_shake);
        mSensroMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        mVibrator = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
    }

    @Override
    protected void onPause() {
        super.onPause();
        mSensroMgr.unregisterListener(this);
    }

    @Override
    protected void onResume() {
        super.onResume();
        mSensroMgr.registerListener(this,
                mSensroMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            // values[0]:x 轴，values[1] ：y 轴，values[2] ：z 轴
            float[] values = event.values;
            if ((Math.abs(values[0]) > 15 || Math.abs(values[1]) > 15 || Math.abs(values[2]) > 15)) {
```

```
                    tv_shake.setText(Utils.getNowTime()+" 恭喜您摇一摇啦");
                    mVibrator.vibrate(500);  //系统检测到摇一摇事件后，震动手机提示用户
                }
            }
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
            //当传感器精度改变时回调该方法，一般无须处理
        }
    }
```

这个例子很简单，一旦监测到手机的摇动幅度超过阈值，就在屏幕上打印摇一摇的结果说明文字，具体效果如图 9-19 所示。



图 9-19　加速度传感器实现简单摇一摇

9.3.3　指南针

顾名思义，指南针只要找到朝南的方向就好了，可是在 App 中并非使用一个方向传感器这么简单，事实上单独的方向传感器已经弃用，取而代之的是利用加速度传感器和磁场传感器，通过 SensorManager 的 getRotationMatrix 方法与 getOrientation 方法计算方向角度。

下面是结合加速度传感器与磁场传感器实现指南针的完整代码：

```
public class DirectionActivity extends AppCompatActivity implements SensorEventListener {
    private TextView tv_direction;
    private CompassView cv_sourth;
    private SensorManager mSensroMgr;
    private float[] mAcceValues;
    private float[] mMagnValues;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_direction);
        tv_direction = (TextView) findViewById(R.id.tv_direction);
        cv_sourth = (CompassView) findViewById(R.id.cv_sourth);
        mSensroMgr = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    }

    @Override
    protected void onPause() {
        super.onPause();
        mSensroMgr.unregisterListener(this);
    }

    @Override
```

```
protected void onResume() {
    super.onResume();
    int suitable = 0;
    List<Sensor> sensorList = mSensroMgr.getSensorList(Sensor.TYPE_ALL);
    for (Sensor sensor : sensorList) {
        if (sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            suitable += 1;
        } else if (sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
            suitable += 10;
        }
    }
    if (suitable/10>0 && suitable%10>0) {
        mSensroMgr.registerListener(this,
                mSensroMgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
                SensorManager.SENSOR_DELAY_NORMAL);
        mSensroMgr.registerListener(this,
                mSensroMgr.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
                SensorManager.SENSOR_DELAY_NORMAL);
    } else {
        cv_sourth.setVisibility(View.GONE);
        tv_direction.setText("当前设备不支持指南针，请检查是否存在加速度和磁场传感器");
    }
}

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
        mAcceValues = event.values;
    } else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
        mMagnValues = event.values;
    }
    if (mAcceValues!=null && mMagnValues!=null) {
        calculateOrientation();
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}

// 计算方向
private void calculateOrientation() {
    float[] values = new float[3];
    float[] R = new float[9];
```

```
            SensorManager.getRotationMatrix(R, null, mAcceValues, mMagnValues);
            SensorManager.getOrientation(R, values);
            values[0] = (float) Math.toDegrees(values[0]);
            cv_sourth.setDirection((int) values[0]);
            if (values[0] >= -10 && values[0] < 10) {
                tv_direction.setText("手机上部方向是正北");
            } else if (values[0] >= 10 && values[0] < 80) {
                tv_direction.setText("手机上部方向是东北");
            } else if (values[0] >= 80 && values[0] <= 100) {
                tv_direction.setText("手机上部方向是正东");
            } else if (values[0] >= 100 && values[0] < 170) {
                tv_direction.setText("手机上部方向是东南");
            } else if ((values[0] >= 170 && values[0] <= 180) || (values[0]) >= -180 && values[0] < -170) {
                tv_direction.setText("手机上部方向是正南");
            } else if (values[0] >= -170 && values[0] < -100) {
                tv_direction.setText("手机上部方向是西南");
            } else if (values[0] >= -100 && values[0] < -80) {
                tv_direction.setText("手机上部方向是正西");
            } else if (values[0] >= -80 && values[0] < -10) {
                tv_direction.setText("手机上部方向是西北");
            }
        }
    }
```

上述代码计算得到的只是手机上部与正北方向的角度，要想在手机上模拟指南针的效果，得自己写一个罗盘视图，然后在罗盘上绘制出正南方向的指针。罗盘视图上的指南针效果如图 9-20 和图 9-21 所示。其中，图 9-20 所示为手机上部对准正南方向的界面，此时指南针恰好位于朝上的方向；转动手机使上部对准正东方向，此时指南针转到了屏幕右边，如图 9-21 所示。

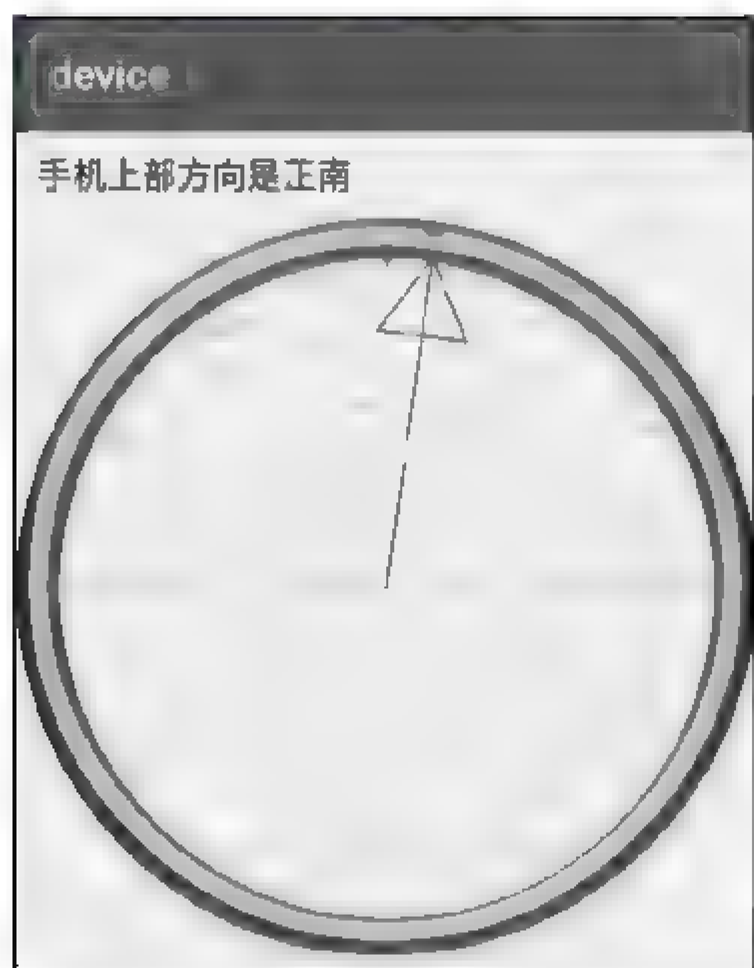


图 9-20　手机上部对准正南方向时的指南针

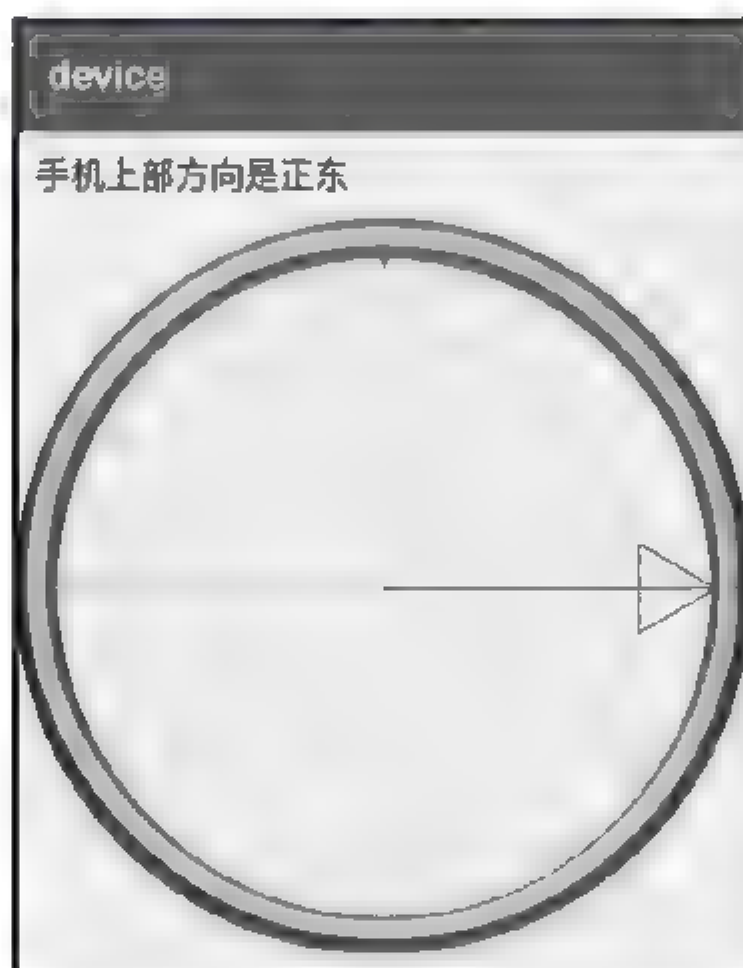


图 9-21　手机上部对准正东方向时的指南针

9.3.4　计步器和感光器

其他传感器各有千秋，合理使用能够产生许多趣味应用。下面分别介绍几款用途较广的例子，包括计步器、感光器等。

1. 计步器

计步器的原理是通过手机的前后摆动模拟步伐节奏的监测。Android 中与计步器有关的传感器有两个，一个是步行检测（TYPE_STEP_DETECTOR），另一个是步行计数（TYPE_STEP_COUNTER）。其中，步行检测的返回数值为 1 时，表示当前监测到一个步伐；步行计数的返回数值是累加后的数值，表示本次开机激活后的总步伐数。

下面是使用计步器的代码片段：

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_STEP_DETECTOR) {
        if (event.values[0] == 1.0f) {
            mStepDetector++;
        }
    } else if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
        mStepCounter = (int) event.values[0];
    }
    String desc = String.format("设备检测到您当前走了%d 步，总计数为%d 步",
            mStepDetector, mStepCounter);
    tv_step.setText(desc);
}
```

计步器的效果如图 9-22 所示，可以看到计步器的总计数是累加值。

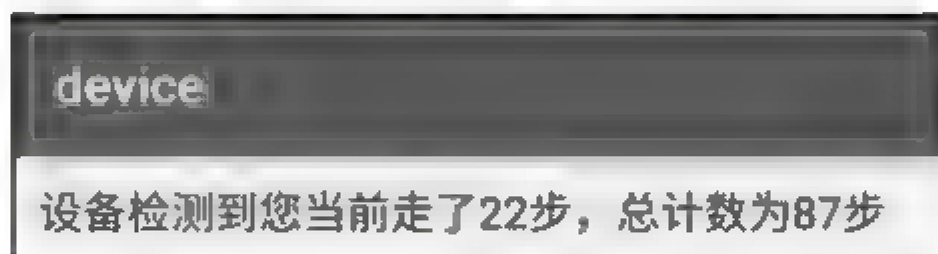


图 9-22　计步器的效果界面

2. 感光器

感光器也叫光线传感器，借助于前置摄像头的曝光，一旦遮住前置摄像头，传感器监测到的光线强度立马就会降低。在实际开发中，光线传感器往往用于感应手机正面的光线强弱，从而自动调节屏幕亮度。

使用光线传感器的代码片段如下：

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_LIGHT) {
        float light_strength = event.values[0];
```

```
                tv_light.setText(Utils.getNowTime() + " 当前光线强度为" + light_strength);
            }
        }
```

光线传感器的效果如图 9-23 所示，光线强度的数值每时每刻都在变化。



图 9-23　光线传感器的效果界面

9.4　手机定位

本节介绍手机定位的手段与实现，首先阐述手机定位的工作原理，接着指出各类定位手段对应的手机功能开关；然后对定位的相关工具类进行详细说明，包括定位条件器 Criteria、定位管理器 LocationManager、定位监听器 LocationListener；最后演示通过定位功能获取定位信息的用法。

9.4.1　开启定位功能

定位功能使用得相当广泛，许多 App 都需要使用定位功能找到用户所在的城市，然后切换到对应的城市频道。根据不同的定位方式，手机定位又分为卫星定位和网络定位两大类。

卫星定位服务由几个全球卫星导航系统提供，主要包括美国 GPS、俄罗斯格洛纳斯、中国北斗。卫星定位的原理是根据多颗卫星与导航芯片的通信结果得到手机与卫星距离，然后计算手机当前所处的经度、纬度以及海拔高度。使用卫星定位需开启手机上的 GPS 功能，并且最好在室外使用。

网络定位有基站定位与 WIFI 定位两个子类。手机插上运营商提供的 SIM 卡后，这个 SIM 卡会搜索周围的基站信号并接入通信服务。手机基站俗称铁塔，每个铁塔都有对应的编号、位置信息、信号覆盖区域。基站定位的原理是监测 SIM 卡能搜索到周围的哪些基站，手机必然处于这些基站信号覆盖的重叠区域，再根据每个基站的位置信息得出手机的大致方位。使用基站定位需开启手机上的数据连接功能。

WIFI 定位的原理是手机接入某个公共热点网络，比如首都机场的 WIFI，提供 WIFI 热点的路由器有自身的 MAC 地址与电信宽带的网络 IP，通过查询 WIFI 路由器的位置便可得知接入该 WIFI 手机的大致位置。使用 WIFI 定位需开启手机上的 WLAN 功能。

无论是基站定位还是 WIFI 定位，手机自身只能获取基站与 WIFI 路由器的信息，无法直接得到手机的位置信息。要想获得具体的方位，必须先把基站或 WIFI 路由器的信息传给位置服务提供商（比如高德地图或百度地图），位置服务器储存了每个基站和 WIFI 路由器的编号、



MAC 地址、实际位置，从这个庞大的网络数据库找到具体基站或 WIFI 的详细位置，再返回给手机客户端。因为需要后端的网络参与计算手机的位置信息，所以基站定位和 WIFI 定位统称为网络定位，国产手机网络定位的计算功能由高德地图和百度地图分别提供。

既然使用这几种定位都要开启对应的手机功能，那么首先得获取这些功能的开关状态，然后根据需要开启或关闭对应的功能。下面是对 GPS、数据连接、WLAN 功能进行状态获取和开关操作的代码：

```
// 获取 GPS 的开关状态
public static boolean getGpsStatus(Context ctx) {
    LocationManager lm = (LocationManager) ctx.getSystemService(Context.LOCATION_SERVICE);
    boolean gps_enabled = lm.isProviderEnabled(LocationManager.GPS_PROVIDER);
    return gps_enabled;
}

// 打开或关闭 GPS
@TargetApi(Build.VERSION_CODES.KITKAT)
public static void setGpsStatus(Context ctx, boolean enabled) {
    Intent gpsIntent = new Intent();
    gpsIntent.setClassName("com.android.settings",
            "com.android.settings.widget.SettingsAppWidgetProvider");
    gpsIntent.addCategory("android.intent.category.ALTERNATIVE");
    gpsIntent.setData(Uri.parse("custom:3"));
    try {
        PendingIntent.getBroadcast(ctx, 0, gpsIntent, 0).send();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// 获取定位的开关状态
public static boolean getLocationStatus(Context ctx) {
    LocationManager lm = (LocationManager) ctx.getSystemService(Context.LOCATION_SERVICE);
    boolean gps_enabled = lm.isProviderEnabled(LocationManager.GPS_PROVIDER);
    boolean network_enabled = lm.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
    return gps_enabled || network_enabled;
}

// 获取 WIFI 的开关状态
public static boolean getWlanStatus(Context ctx) {
    WifiManager wm = (WifiManager) ctx.getSystemService(Context.WIFI_SERVICE);
    return wm.isWifiEnabled();
}
```

```
    // 打开或关闭 WIFI
    public static void setWlanStatus(Context ctx, boolean enabled) {
        WifiManager wm = (WifiManager) ctx.getSystemService(Context.WIFI_SERVICE);
        wm.setWifiEnabled(enabled);
    }

    // 获取数据连接的开关状态
    public static boolean getMobileDataStatus(Context ctx) {
        ConnectivityManager cm = (ConnectivityManager) ctx.getSystemService(Context.
CONNECTIVITY_SERVICE);
        boolean isOpen = false;
        try {
            Method method = cm.getClass().getMethod("getMobileDataEnabled");
            isOpen = (Boolean) method.invoke(cm);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return isOpen;
    }

    // 打开或关闭数据连接
    public static void setMobileDataStatus(Context ctx, boolean enabled) {
        ConnectivityManager cm = (ConnectivityManager) ctx.getSystemService(Context.
CONNECTIVITY_SERVICE);
        try {
            Method method = cm.getClass().getMethod("setMobileDataEnabled", Boolean.TYPE);
            method.invoke(cm, enabled);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

以上定位的相关功能还需在 AndroidManifest.xml 中补充对应的权限信息，具体的权限说明如下：

```
    <!-- 定位 -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <!-- 查看网络状态 -->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <!-- 查看手机状态 -->
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

9.4.2 获取定位信息

开启定位相关功能只是将定位的前提条件准备好，若想获得手机当前所处的位置信息，还要依靠一系列定位工具。与定位信息获取有关的工具有定位条件器 Criteria、定位管理器 LocationManager、定位监听器 LocationListener。下面对这 3 个工具分别进行介绍。

1. 定位条件器 Criteria

定位条件器用于设置定位的前提条件，比如精度、速度、海拔、方位等信息，有以下 4 个常用参数。

- setAccuracy：设置定位精确度。有两个取值，Criteria.ACCURACY_FINE 表示精度高，Criteria.ACCURACY_COARSE 表示精度低。
- setSpeedAccuracy：设置速度精确度。速度精确度的取值说明见表 9-8。

表9-8 速度精确度的取值说明

Criteria 类的速度精确度	说明
ACCURACY_HIGH	精度高，误差小于 100 米
ACCURACY_MEDIUM	精度中等，误差在 100 米到 500 米之间
ACCURACY_LOW	精度低，误差大于 500 米

- setAltitudeRequired：设置是否需要海拔信息。取值 true 表示需要，false 表示不需要。
- setBearingRequired：设置是否需要方位信息。取值 true 表示需要，false 表示不需要。
- setCostAllowed：设置是否允许运营商收费。取值 true 表示允许，false 表示不允许。
- setPowerRequirement：设置对电源的需求。有 3 个取值，Criteria.POWER_LOW 表示耗电低，Criteria.POWER_MEDIUM 表示耗电中等，Criteria.POWER_HIGH 表示耗电高。

2. 定位管理器 LocationManager

定位管理器用于获取定位信息的提供者、设置监听器，并获取最近一次的位置信息。定位管理器的对象从系统服务 LOCATION_SERVICE 获取，常用方法有以下 7 个。

- getBestProvider：获取最佳的定位提供者。第一个参数为定位条件器 Criteria 的实例，第二个参数取值 true 表示只要可用的。定位提供者的取值说明见表 9-9。

表9-9 定位提供者的取值说明

定位提供者的名称	说明	定位功能的开启状态
gps	卫星定位	开启 GPS 功能
network	网络定位	开启数据连接或 WLAN 功能
passive	无法定位	未开启定位相关功能

- isProviderEnabled：判断指定的定位提供者是否可用。
- getLastKnownLocation：获取最近一次的定位地点。

- requestLocationUpdates：设置定位监听器。其中，第一个参数为定位提供者，第二个参数为位置更新的最小间隔时间，第三个参数为位置更新的最小距离，第四个参数为定位监听器实例。
- removeUpdates：移除定位监听器。
- addGpsStatusListener：添加定位状态的监听器。该监听器需实现 GpsStatus.Listener 接口的 onGpsStatusChanged 方法。
- removeGpsStatusListener：移除定位状态的监听器。

3. 定位监听器 LocationListener

定位监听器用于监听定位信息的变化事件，如定位提供者的开关、位置信息发生变化等。该监听器可使用以下 4 种方法。

- onLocationChanged：在位置地点发生变化时调用。在此可获取最新的位置信息。
- onProviderDisabled：在定位提供者被用户关闭时调用。
- onProviderEnabled：在定位提供者被用户开启时调用。
- onStatusChanged：在定位提供者的状态变化时调用。定位提供者的状态取值说明见表 9-10。

表9-10 定位提供者的状态取值说明

LocationProvider 类的状态类型	说明
OUT_OF_SERVICE	在服务范围外
TEMPORARILY_UNAVAILABLE	暂时不可用
AVAILABLE	可用状态

获取定位信息的示例代码如下：

```
public class LocationActivity extends AppCompatActivity {
    private final static String TAG = "LocationActivity";
    private TextView tv_location;
    private String mLocation="";
    private LocationManager mLocationMgr;
    private Criteria mCriteria = new Criteria();
    private Handler mHandler = new Handler();
    private boolean bLocationEnable = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_location);
        initWidget();
        initLocation();
        mHandler.postDelayed(mRefresh, 100);
```



```
    }

    private void initWidget() {
        tv_location = (TextView) findViewById(R.id.tv_location);
        mLocationMgr = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        mCriteria.setAccuracy(Criteria.ACCURACY_FINE);  // 设置定位精确度
        mCriteria.setAltitudeRequired(true);  // 设置是否需要海拔信息 Altitude
        mCriteria.setBearingRequired(true);  // 设置是否需要方位信息 Bearing
        mCriteria.setCostAllowed(true);  // 设置是否允许运营商收费
        mCriteria.setPowerRequirement(Criteria.POWER_LOW);  // 设置对电源的需求
    }

    private void initLocation() {
        String bestProvider = mLocationMgr.getBestProvider(mCriteria, true);
        if (bestProvider == null) {
            bestProvider = LocationManager.NETWORK_PROVIDER;
        }
        if (mLocationMgr.isProviderEnabled(bestProvider)) {
            tv_location.setText("正在获取"+bestProvider+"定位对象");
            mLocation = String.format("定位类型=%s", bestProvider);
            beginLocation(bestProvider);
            bLocationEnable = true;
        } else {
            tv_location.setText("\n"+bestProvider+"定位不可用");
            bLocationEnable = false;
        }
    }

    private void setLocationText(Location location) {
        if (location != null) {
            String desc = String.format("%s\n 定位对象信息如下 ：  " +
                    "\n\t 其中时间 ： %s" + "\n\t 其中经度 ： %f，纬度 ： %f" +
                    "\n\t 其中高度 ： %d 米，精度 ： %d 米",
                    mLocation, Utils.getNowDateTimeFormat(),
                    location.getLongitude(), location.getLatitude(),
                    Math.round(location.getAltitude()), Math.round(location.getAccuracy()));
            Log.d(TAG, desc);
            tv_location.setText(desc);
        } else {
            tv_location.setText(mLocation+"\n 暂未获取定位对象");
        }
    }
```

```
    private void beginLocation(String method) {
        mLocationMgr.requestLocationUpdates(method, 300, 0, mLocationListener);
        Location location = mLocationMgr.getLastKnownLocation(method);
        setLocationText(location);
    }

    // 位置监听器
    private LocationListener mLocationListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            setLocationText(location);
        }

        @Override
        public void onProviderDisabled(String arg0) {
        }

        @Override
        public void onProviderEnabled(String arg0) {
        }

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle arg2) {
        }
    };

    private Runnable mRefresh = new Runnable() {
        @Override
        public void run() {
            if (bLocationEnable == false) {
                initLocation();
                mHandler.postDelayed(this, 1000);
            }
        }
    };

    @Override
    protected void onDestroy() {
        if (mLocationMgr != null) {
            mLocationMgr.removeUpdates(mLocationListener);
        }
        super.onDestroy();
    }
}
```

获取定位信息的效果如图 9-24 所示，当前定位类型是卫星定位，定位结果是东经 119 度、北纬 26 度，海拔高度 77 米，定位精度 6 米。



图 9-24　某设备获取的定位信息

9.5　实战项目：仿微信的发现功能

本章涉及的知识点比较庞杂，前面介绍的大多是基础功能应用，很少有实际业务的使用说明。本节的实战项目谈谈手机的设备功能在商用 App 中的具体应用，让读者站在用户角度对设备操作有一个感性认识，如果想做一个受欢迎的 App，不仅需要钻研技术，更要贴近用户生活，研发易用、好用、值得用的 App。

9.5.1　设计思路

微信的用户量巨大，不少小功能都很人性化，比如发现频道，如图 9-25 所示。

发现频道提供的小功能包括扫一扫、摇一摇、附近的人、漂流瓶等，如附近的人、漂流瓶等需要多人参与的功能不纳入本次实战项目，扫一扫与摇一摇相对纯粹，加入实战项目当中。

另外，支付宝原来有一个咻一咻功能也很有名。2016 年前的除夕夜，全民开启咻一咻疯抢五福卡，这个场景片段登上了当年的春晚荧屏。不过，支付宝日常的咻一咻并非寻找福袋，而是搜索用户附近的商家信息。

现在，我们综合微信与支付宝的几个小功能组成本章的实战项目——“仿微信的发现功能”。该功能内含 3 个模块，分别是扫一扫、摇一摇和咻一咻，入口效果如图 9-26 所示。



图 9-25　微信的发现频道截图



图 9-26　仿微信的发现功能页面截图

下面分别说明这 3 个模块将要实现的功能。

1. 扫一扫（扫描二维码）

该模块与微信的“扫一扫”基本类似，通过扫描二维码图片识别二维码携带的字符串信息。Android 中的二维码扫描可用谷歌的 zxing 工具包结合 zxing 的开源框架完成扫码与识别操作。扫一扫的效果如图 9-27 所示，此时手机在进行图像识别。

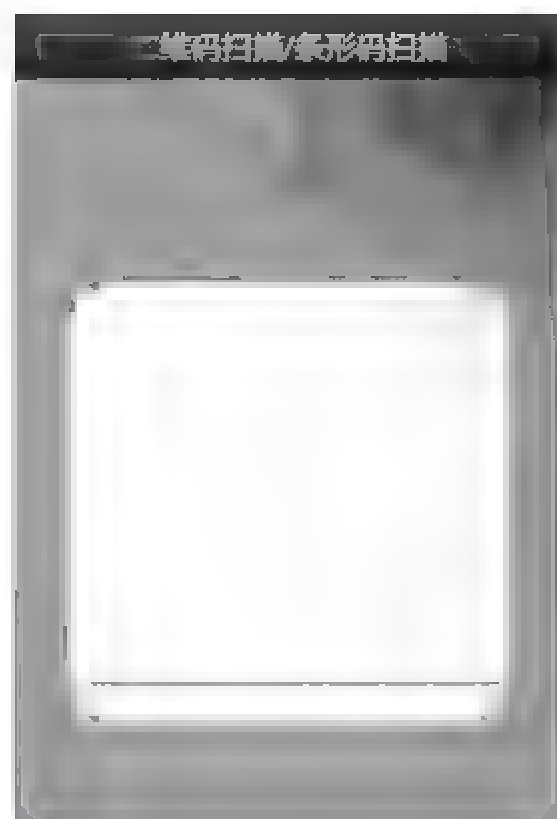


图 9-27　扫一扫（扫描二维码）的初始界面

2. 摇一摇（博饼抽大奖）

微信的“摇一摇”可以摇人、摇歌曲、摇电视，我们另辟蹊径，做一个摇骰子的游戏——“中秋博饼”。300 多年前，民族英雄郑成功率军驻扎在厦门进行抗清活动，每逢中秋佳节，为宽慰士兵的思乡之情，发明了名为“博饼”的摇骰子游戏，依据不同的幸运点数判定不同的中奖级别。经过几百年的流传，中秋节博饼的习俗已经广泛流传于闽台两地与东南亚。本实战项目通过摇手机触发摇骰子的动作，进而计算每次的中奖结果。博饼抽大奖的效果如图 9-28 所示，这是博饼游戏的初始界面。

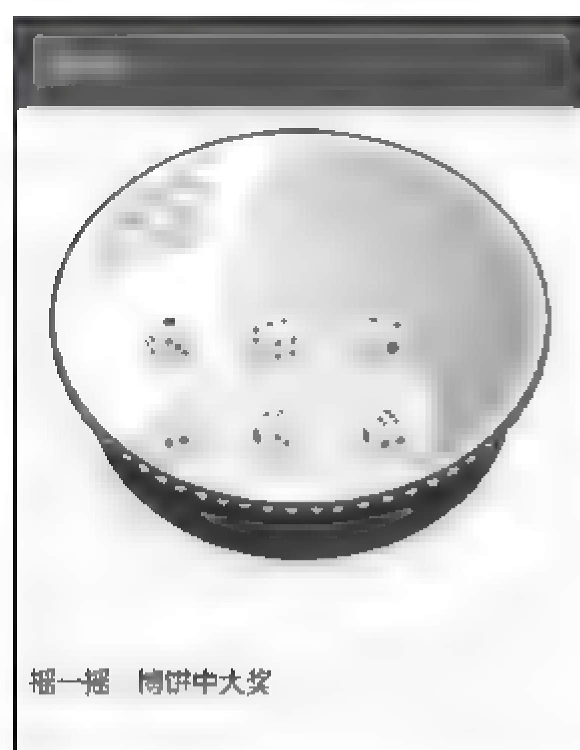


图 9-28　摇一摇（博饼抽大奖）的初始界面



图 9-29　咻一咻（卫星浑天仪）的初始界面

3. 咻一咻（卫星浑天仪）

浑天仪为东汉科学家张衡发明，用于观测天象，日月星辰皆可在浑天仪上找到对应的位置。随着现代科技的发展，我们已经不满足于自古以来就有的日月星辰，而是把现在的科技成果展示出来。前面提到，手机定位的一大手段是卫星定位，既然卫星能够发现手机的位置，反过来手机也能发现卫星的方位，把手机（导航芯片）监测到的卫星逐个标记在罗盘上岂不构成了一个卫星浑天仪？卫星浑天仪的效果如图 9-29 所示，一开始只有一个罗盘，具体的卫星信息还有待在代码中获取，并显示到罗盘上。

分析实战项目 3 个模块的功能大致包含本章哪些知识点？读者肯定能找到以下 4 点。

（1）相机类 Camera：扫描二维码需要摄像头支持，必然用到 Camera。

（2）加速度传感器SensorManager：前面介绍加速度传感器时已经提到它通常用于摇一摇。



（3）卫星定位 LocationManager：无论是根据卫星找手机的位置，还是通过手机监测卫星的位置，都会用到定位功能。

（4）媒体播放 MediaPlayer：好几个场景需要播放声音，比如二维码识别完毕的“哔”声，摇一摇的摇骰子声，咻一咻的“咻咻”声，这些都要使用 MediaPlayer 播音。

涉及的知识点不算多，但也基本涵盖了每节的代表技术。

9.5.2 小知识：全球卫星导航系统

卫星导航是高科技的航天技术，目前联合国认可的全球卫星导航系统有 4 个，分别是美国的 GPS、俄罗斯的格洛纳斯、中国的北斗和欧洲的伽利略，其中真正投入商用的只有前 3 个。

（1）美国的 GPS：GPS 是 Global Positioning System（全球定位系统）的简称，于 1964 年投入使用。到 1993 年，包含 24 颗卫星的 GPS 系统完成组网。

（2）俄罗斯的格洛纳斯：格洛纳斯（GLONASS）是俄语对全球卫星导航系统 Global Navigation Satellite System 的简称，该系统于 2007 年开始运营，并在 2011 年完成 24 颗卫星的组网。

（3）中国的北斗：北斗（BeiDou Navigation Satellite System，BDS）是中国自行研制的全球卫星导航系统，是继美国 GPS、俄罗斯格洛纳斯之后第 3 个成熟的卫星导航系统。北斗在 2007 年开始提供定位服务，2012 年完成 16 颗卫星的亚太地区组网，计划于 2020 年完成 35 颗卫星的全球组网。

（4）欧洲的伽利略：伽利略卫星导航系统（Galileo Satellite Navigation System）是由欧盟研制和建立的全球卫星导航定位系统，于 2013 年完成 4 颗卫星的初步组网，但建设进度严重滞后，能够提供定位服务仍是遥遥无期。

目前，智能手机一般都内置 GPS 的导航芯片，只有部分中、高端手机同时内置格洛纳斯与北斗的导航芯片。

要想获取天上的卫星信息，得调用定位管理器 LocationManager 对象的 addGpsStatusListener 方法添加定位状态监听器，该监听器需实现 GpsStatus.Listener 接口的 onGpsStatusChanged 方法，该方法提供了定位状态变化的事件信息，事件类型的取值说明见表 9-11。

表9-11 GPS事件类型的取值说明

GpsStatus 类的事件类型	说明
GPS_EVENT_STARTED	GPS 功能开启
GPS_EVENT_STOPPED	GPS 功能停止
GPS_EVENT_FIRST_FIX	首次定位
GPS_EVENT_SATELLITE_STATUS	周期地报告卫星状态

其中，最后一个卫星状态报告事件可以获得监测到的卫星信息，一旦捕获该事件，即可调用 LocationManager 对象的 getGpsStatus 方法获得当前的定位状态信息 GpsStatus，再调用 GpsStatus 对象的 getSatellites 方法获得本次监测到的卫星列表，卫星列表是一个 GpsSatellite 队列，详细的卫星信息可通过 GpsSatellite 对象的以下方法获得。

- getPrn：卫星的伪随机码，可以认为是卫星的编号。
- getAzimuth：获取卫星的方位角。
- getElevation：获取卫星的仰角。
- getSnr：卫星的信噪比，即信号强弱。
- hasAlmanac：卫星是否有年历表。
- hasEphemeris：卫星是否有星历表。
- usedInFix：卫星是否被用于近期的 GPS 修正计算。

在这些信息中，对确定卫星位置有用的主要有 3 个，分别是卫星编号（用于确定卫星的国籍）、卫星的方位角（用于确定卫星的方向）和卫星的仰角（用于确定卫星的远近距离）。

下面是获取导航卫星信息的监听器代码片段：

```
private GpsStatus.Listener mStatusListener = new GpsStatus.Listener() {
    @Override
    public void onGpsStatusChanged(int event) {
        GpsStatus gpsStatus = mLocationMgr.getGpsStatus(null);
        switch (event) {
        case GpsStatus.GPS_EVENT_FIRST_FIX:
            break;
        case GpsStatus.GPS_EVENT_SATELLITE_STATUS: // 周期的报告卫星状态
            // 得到所有卫星信息，包括高度角、方位角、信噪比和伪随机号（卫星编号）
            Iterable<GpsSatellite> satellites = gpsStatus.getSatellites();
            for (GpsSatellite satellite : satellites) {
                Satellite item = new Satellite();
                item.seq = satellite.getPrn();
                item.signal = Math.round(satellite.getSnr());
                item.elevation = Math.round(satellite.getElevation());
                item.azimuth = Math.round(satellite.getAzimuth());
                item.time = Utils.getNowDateTime();
                if (item.seq <= 64 || (item.seq >= 120 && item.seq <= 138)) {
                    item.nation = "美国";
                    item.name = "GPS";
                } else if (item.seq >= 201 && item.seq <= 237) {
                    item.nation = "中国";
                    item.name = "北斗";
                } else if (item.seq >= 65 && item.seq <= 89) {
                    item.nation = "俄罗斯";
                    item.name = "格洛纳斯";
                } else {
                    item.nation = "其他";
                    item.name = "未知";
                }
                mapSatellite.put(item.seq, item);
```



```
                }
                cv_satellite.setSatelliteMap(mapSatellite);
            case GpsStatus.GPS_EVENT_STARTED:
                break;
            case GpsStatus.GPS_EVENT_STOPPED:
                break;
            default:
                break;
            }
        }
    };
```

9.5.3 代码示例

从本章开始，代码示例一节将更侧重于在真机上进行相关测试，对于编码上的说明仅限于要注意或容易遗漏的地方。编码与测试方面需要注意以下5点：

（1）扫一扫用到了zxing工具包，要在libs目录导入对应的JAR包，即zxing3.2.1.jar。同时还要在Java源码目录导入com.app.zxing的开源框架。

（2）使用摄像头与定位功能，不要忘了往AndroidManifest.xml添加对应的权限配置。

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

（3）在res/raw目录下保存播放"哔"声的音频文件，在res/values目录下保存zxing框架依赖的ids.xml。

（4）需要自定义一个博饼视图BettingView，用于展示摇骰子的动态效果。还需自定义一个罗盘视图CompassView，用于展示天空坐标和天上的卫星分布图。

（5）App需要完全的GPS权限，不需要提示那种，否则打开用到GPS功能的页面时会闪退。

实战项目在模拟器上测试通过后，按照第8章的说明将App安装到手机上，使用真机进行实际的功能测试。

首先测试扫一扫功能。图9-30所示为一张清华大学微信公众号的二维码图片。扫描结果如图9-31所示，识别的字符串是一个指向微信服务器的HTTP连接字符串。



图9-30 清华大学微信公众号的二维码图片



图9-31 扫描二维码的识别结果

扫一扫其实不只可以扫描二维码，还可扫描条形码。图 9-32 所示为一张常见的商品条形码图片。扫描结果如图 9-33 所示，识别的是一串数字编号。



图 9-32　某商品的条形码图片



图 9-33　扫描条形码的识别结果

接着测试摇一摇功能，拿起手机使劲晃荡几下，看看屏幕界面是不是动了起来？图 9-34 与图 9-35 是两张不同的中奖效果图。其中，图 9-34 表示摇中了秀才奖（一个红四），图 9-35 表示摇中了状元奖（4 个红四）。

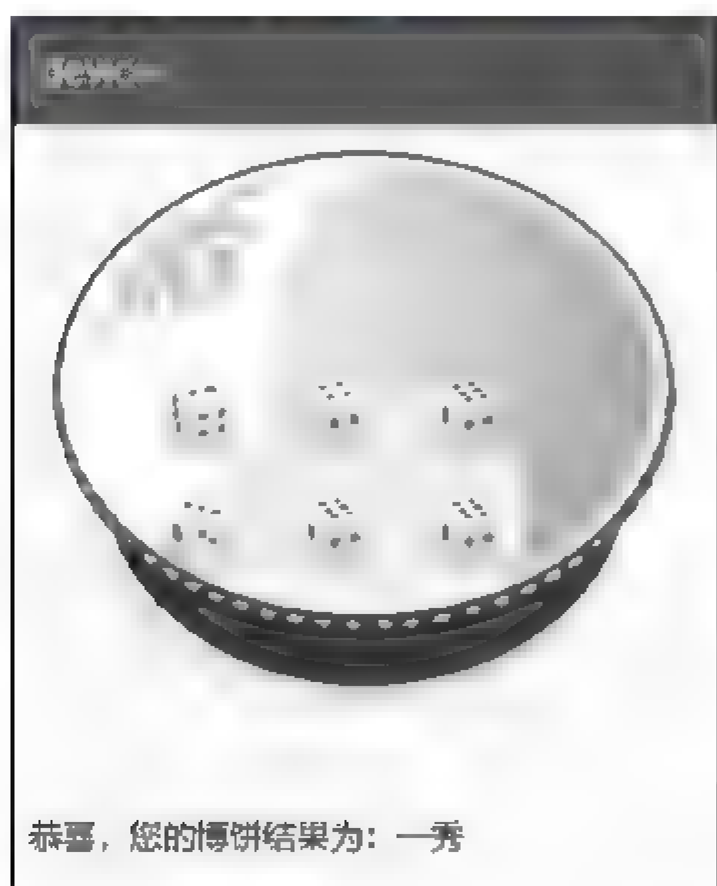


图 9-34　摇一摇中了一秀

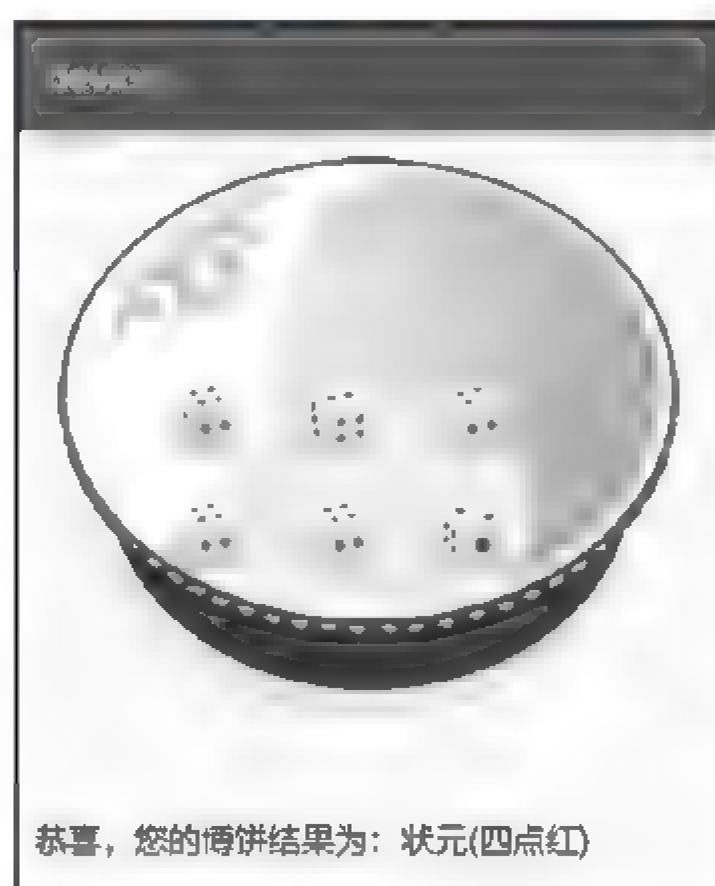


图 9-35　摇一摇中了状元

因为骰子上的四点与一点为红色，所以博饼的中奖点数围绕红四与红一制定。下面来看具体的中奖规则。

状元插金花：4 个红四加两个红一
六杯红：有 6 个红四
遍地锦：有 6 个红一
五红：有 5 个红四
四点红：有 4 个红四
五子登科：有 5 个相同的点数（5 个红四除外）

上面几个都是状元，以状元插金花为最大。

对堂（榜眼和探花）：6 个骰子分别是一、二、三、四、五、六
四进（进士）：有 4 个相同的点数（4 个红四除外）
三红：有 3 个红四
二举（举人）：有两个红四
一秀（秀才）：只有一个红四



中国幅员辽阔，各地都有自己的风俗，骰子也不例外，不知道读者当地的骰子是什么玩法，要不要把你们的玩法写到摇一摇里面去呢？

最后测试咻一咻功能，这个必须找个好一点的手机，因为配置好的手机才有格洛纳斯与北斗的导航芯片。图 9-36 所示的手机只支持 GPS 芯片，效果图上满屏都是美国的卫星。图 9-37 所示的手机同时内置 GPS、格洛纳斯与北斗的芯片，效果图上的卫星三国都有。

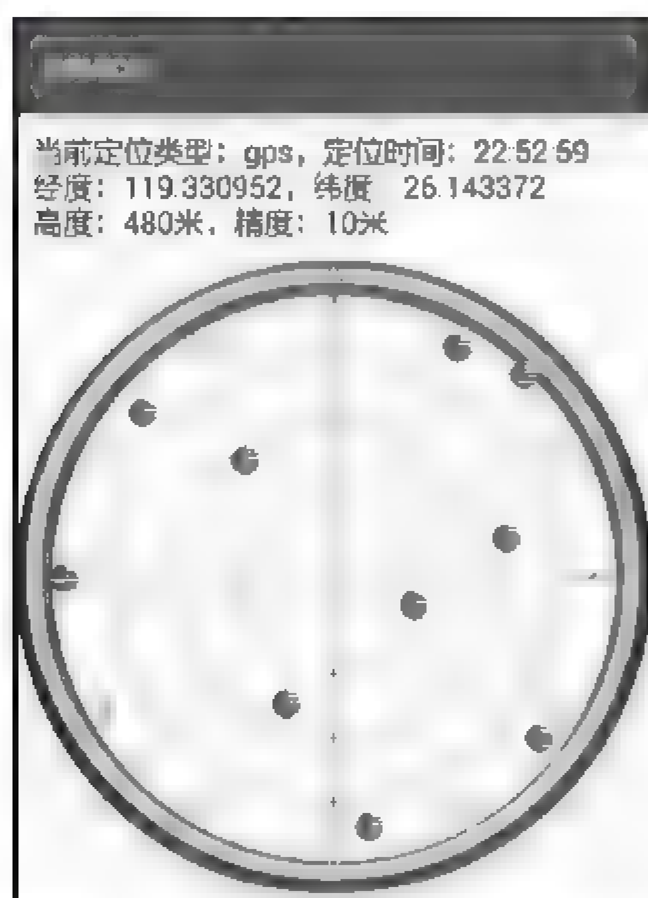


图 9-36　只支持 GPS 的卫星分布图



图 9-37　支持 3 种导航系统的卫星分布图

如果手机只支持 GPS，定位响应就很慢，定位精度一般在 10 米左右，而且定位高度很不准，误差相当大。一旦有北斗与格洛纳斯参与定位，即使在室内也能很快响应，精度一般能提升至 5 米，并且高度数值准确了许多，特别适合亚太地区的定位需求。

再来看图 9-37 所示的卫星分布。在笔者头顶这片天空半小时内一共找到 11 颗 GPS 卫星、6 颗格洛纳斯卫星、12 颗北斗卫星，原来中国的北斗已经赶上并超过美国的 GPS 了。身为中国人的你，有没有感到无比感动与自豪？快快拿出手机试试咻一咻功能，看看你头上的天空能找到几颗卫星。

下面是罗盘视图 CompassView 的代码（可被指南针与浑天仪两个功能复用）：

```
public class CompassView extends View {
    private final static String TAG = "CompassView";
    private Context mContext;
    private int mWidth;
    private Paint mPaintLine;
    private Paint mPaintText;
    private Paint mPaintAngle;
    private Bitmap mCompassBg;
    private Rect mRectSrc;
    private Rect mRectDest;
    private RectF mRectAngle;
    private RectF mRectSourth;
    private Bitmap mSatelliteChina;
```

```
    private Bitmap mSatelliteAmerica;
    private Bitmap mSatelliteRussia;
    private Map<Integer, Satellite> mapSatellite = new HashMap<Integer, Satellite>();
    private int mScaleLength = 25;
    private float mBorder = 0.9f;
    private int mDirection = -1024;
    private Paint mPaintSourth;

    public CompassView(Context context) {
        this(context, null);
    }

    public CompassView(Context context, AttributeSet attr) {
        super(context, attr);
        mContext = context;
        mPaintLine = new Paint();
        mPaintLine.setAntiAlias(true);
        mPaintLine.setColor(Color.GREEN);
        mPaintLine.setStrokeWidth(2);
        mPaintLine.setStyle(Style.STROKE);
        mPaintText = new Paint();
        mPaintText.setAntiAlias(true);
        mPaintText.setColor(Color.RED);
        mPaintText.setStrokeWidth(1);
        mPaintText.setStyle(Style.STROKE);
        mPaintText.setTextSize(28);
        mPaintAngle = new Paint();
        mPaintAngle.setAntiAlias(true);
        mPaintAngle.setColor(Color.GRAY);
        mPaintAngle.setStrokeWidth(1);
        mPaintAngle.setStyle(Style.STROKE);
        mPaintAngle.setTextSize(23);
        mPaintSourth = new Paint();
        mPaintSourth.setAntiAlias(true);
        mPaintSourth.setColor(Color.RED);
        mPaintSourth.setStrokeWidth(4);
        mPaintSourth.setStyle(Style.STROKE);
        mCompassBg = BitmapFactory.decodeResource(getResources(), R.drawable.compass_bg);
        mRectSrc = new Rect(0, 0, mCompassBg.getWidth(), mCompassBg.getHeight());
        mSatelliteChina = BitmapFactory.decodeResource(getResources(), R.drawable.satellite_china);
        mSatelliteAmerica = BitmapFactory.decodeResource(getResources(), R.drawable.satellite_america);
        mSatelliteRussia = BitmapFactory.decodeResource(getResources(), R.drawable.satellite_russia);
    }

    @Override
```



```
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int width = View.MeasureSpec.getSize(widthMeasureSpec);
        int height = View.MeasureSpec.getSize(heightMeasureSpec);
        mWidth = getMeasuredWidth();
        if (width < height) {
                super.onMeasure(widthMeasureSpec, widthMeasureSpec);
        } else {
                super.onMeasure(heightMeasureSpec, heightMeasureSpec);
        }
        mRectDest = new Rect(0, 0, mWidth, mWidth);
        mRectAngle = new RectF(mWidth/10, mWidth/10, mWidth*9/10, mWidth*9/10);
        mRectSourth = new RectF(mWidth*0.3f/10, mWidth*0.3f/10, mWidth*9.7f/10, mWidth*9.7f/10);
        Log.d(TAG, "mWidth="+mWidth);
}

@Override
protected void dispatchDraw(Canvas canvas) {
        super.dispatchDraw(canvas);
        int radius = mWidth/2;
        int margin = radius/10;
        canvas.drawBitmap(mCompassBg, mRectSrc, mRectDest, new Paint());
        canvas.drawCircle(radius, radius, radius*3/10, mPaintLine);
        canvas.drawCircle(radius, radius, radius*5/10, mPaintLine);
        canvas.drawCircle(radius, radius, radius*7/10, mPaintLine);
        canvas.drawCircle(radius, radius, radius*9/10, mPaintLine);
        canvas.drawLine(radius, margin, radius, mWidth-margin, mPaintLine);
        canvas.drawLine(margin, radius, mWidth-margin, radius, mPaintLine);
        //画罗盘的刻度
        for (int i=0; i<360; i+=30) {
                Path path = new Path();
                path.addArc(mRectAngle, i-3, i+3);
                int angle = (i+90)%360;
                canvas.drawTextOnPath(""+angle, path, 0, 0, mPaintAngle);
                canvas.drawLine(getXpos(radius, angle, radius*mBorder),
                                getYpos(radius, angle, radius*mBorder),
                                getXpos(radius, angle, (radius-mScaleLength)*mBorder),
                                getYpos(radius, angle, (radius-mScaleLength)*mBorder), mPaintAngle);
        }
        //画卫星分布图
        for (Map.Entry<Integer, Satellite> item_map : mapSatellite.entrySet()) {
                Satellite item = item_map.getValue();
                Bitmap bitmap;
                if (item.nation.equals("中国")) {
                        bitmap = mSatelliteChina;
```

```
                } else if (item.nation.equals("美国")) {
                    bitmap = mSatelliteAmerica;
                } else if (item.nation.equals("俄罗斯")) {
                    bitmap = mSatelliteRussia;
                } else {
                    continue;
                }
                float left = getXpos(radius, item.azimuth, radius*mBorder*getCos(item.elevation));
                float top = getYpos(radius, item.azimuth, radius*mBorder*getCos(item.elevation));
                canvas.drawBitmap(bitmap, left-bitmap.getWidth()/2, top-bitmap.getHeight()/2, new Paint());
            }
            //画指南针
            if (mDirection > -1024) {
                int angle = (-mDirection+450)%360;
                canvas.drawLine(getXpos(radius, angle, radius*mBorder),
                        getYpos(radius, angle, radius*mBorder),
                        getXpos(radius, angle, 0), getYpos(radius, angle, 0),mPaintSourth);
                canvas.drawLine(getXpos(radius, angle, radius*mBorder),
                        getYpos(radius, angle, radius*mBorder),
                        getXpos(radius, angle-10, radius*7/10), getYpos(radius, angle-10, radius*7/10),
                        mPaintSourth);
                canvas.drawLine(getXpos(radius, angle, radius*mBorder),
                        getYpos(radius, angle, radius*mBorder),
                        getXpos(radius, angle+10, radius*7/10), getYpos(radius, angle+10, radius*7/10),
                        mPaintSourth);
                canvas.drawLine(getXpos(radius, angle-10, radius*7/10),
                        getYpos(radius, angle-10, radius*7/10),
                        getXpos(radius, angle+10, radius*7/10), getYpos(radius, angle+10, radius*7/10),
                        mPaintSourth);
                Path path = new Path();
                path.addArc(mRectSourth, angle-2, angle+2);
                canvas.drawTextOnPath("南", path, 0, 0, mPaintText);
            } else {
                canvas.drawText("北", radius-15, margin-15, mPaintText);
            }
        }

        private float getXpos(int radius, int angle, double length) {
            return (float) (radius + getCos(angle) * length);
        }

        private float getYpos(int radius, int angle, double length) {
            return (float) (radius + getSin(angle) * length);
        }
```

```
    private double getSin(int angle) {
        return Math.sin(Math.PI*angle/180.0);
    }

    private double getCos(int angle) {
        return Math.cos(Math.PI*angle/180.0);
    }

    public void setSatelliteMap(Map<Integer, Satellite> map) {
        mapSatellite = map;
        invalidate();
    }

    public void setDirection(int direction) {
        mDirection = direction;
        invalidate();
    }
}
```

9.6 小　结

本章主要阐述了手机上硬件设备的使用介绍与操作说明，包括摄像头的用法（表面视图、相机、纹理视图、升级版相机）、麦克风的用法（拖动条、音量控制、录音与播音、录像与放映）、传感器的用法（传感器的种类、加速度传感器、指南针、计步器和感光器）、手机定位的用法（定位的原理、开启定位功能、获取定位信息）。最后设计了一个实战项目“仿微信的发现功能”，在该项目的 App 编码中，实现了扫一扫（扫描二维码）、摇一摇（博饼抽大奖）、咻一咻（卫星浑天仪）3 种功能。另外，介绍了卫星导航的相关知识。

通过本章的学习，读者应能掌握以下 5 种开发技能：

（1）学会操纵相机实现拍照功能（含单拍和连拍）。
（2）学会操纵相机与麦克风实现媒体录制功能（含录音和录像）。
（3）学会音频和视频的播放功能。
（4）学会常见传感器的用法（含加速度传感器、磁场传感器、计步传感器等）。
（5）学会如何获取位置信息（含卫星定位和网络定位）。





网络通信

本章介绍 App 开发常用的一些网络通信技术，主要包括如何使用多线程完成异步操作、如何进行 HTTP 接口调用与图片获取、如何实现文件上传和下载操作、如何运用 Socket 通信技术等。最后结合本章所学的知识演示一个实战项目“仿手机 QQ 的聊天功能”的设计与实现。

10.1 多 线 程

本节介绍多线程技术在App开发中的具体运用，首先说明如何利用Message配合Handler完成主线程与分线程之间的简单通信；然后阐述进度对话框的用法，以及如何自定义实现文本进度条与文本进度圈；接着讲述异步任务AsyncTask的具体用法和注意事项；最后分析异步服务IntentService的实现原理和开发步骤。

10.1.1 消息传递Message

为了使App运行得更流畅，多线程技术被广泛应用于App开发。由于Android系统存在限制，只有主线程才能直接操作界面，因此分线程想修改界面就得另想办法。第9章在介绍摄像头拍照时提到为了让分线程能够刷新界面，Android专门设计了表面视图SurfaceView给分线程操作，后来又增加了纹理视图TextureView，也是给分线程使用。

多线程技术并非单单用于拍照预览，还用于网络通信、后台服务等耗时场合，并且这些场合往往希望操纵现有的界面，而不是操纵表面视图。这要求有一种用于线程之间相互通信的机制。大家都知道，主线程向分线程传递消息时可以直接在分线程的构造函数中传递参数，然而分线程向主线程传递消息并无捷径，为此Android设计了一个Message消息工具，通过结合Handler与Message可简单有效地实现线程之间的通信。

主线程与分线程之间传递消息的步骤主要有4步，说明如下：

1. 在主线程中构造一个Handler对象，并启动分线程

处理器Handler是大家的老朋友了，从第2章开始，凡是需要进行延迟处理的场合，基本都用到了Handler。特别是在第6章，在介绍简单动画的实现时还专门对Handler+Runnable组合做了详细说明。Thread类是Runnable接口的一个具体实现，Handler调用Runnable对象的各种post方法也适用于Thread对象。启动分线程有两种方式，既可通过Handler对象的post方法启动Thread，也可直接调用Thread对象的start方法。

2. 在分线程中构造一个Message对象的消息包

Message是多线程通信中存放消息的包裹，作用类似于Intent机制的Bundle工具。实例可通过自身的obtain方法获得，也可通过Handler对象的obtainMessage方法获得。

下面来看Message类的主要参数说明。

- what：整型的消息标识，用于标识本次消息的唯一编号。
- arg1：整型数，可存放消息的处理结果。
- arg2：整型数，可存放消息的处理代码。
- obj：Object类型，可存放返回消息的数据结构。
- replyTo：Messenger类型，回应信使，在跨进程通信中使用，多线程通信用不着。

3. 在分线程中通过 Handler 对象将 Message 消息发出去

处理器 Handler 的消息发送操作主要是各类 send 方法。下面介绍相关方法说明。

- obtainMessage：获取当前消息的对象。
- sendMessage：立即发送消息。
- sendMessageDelayed：延迟一段时间后发送消息。
- sendMessageAtTime：在指定时间点发送消息。
- sendEmptyMessage：立即发送空消息。
- sendEmptyMessageDelayed：延迟一段时间后发送空消息。
- sendEmptyMessageAtTime：在指定时间点发送空消息。
- removeMessages：从消息队列中根据指定标识移除对应消息。
- hasMessages：判断消息队列中是否存在指定标识的消息。

4. 主线程中的 Handler 对象处理接收到的消息

主线程处理分线程发出的消息需要实现 Handler 对象的 handleMessage 方法，根据 Message 消息的具体内容分别进行相应处理。注意，因为 handleMessage 方法处于主线程（UI 线程）中，所以该方法内部可以直接操作界面元素。

下面是利用多线程实现新闻滚动的完整代码，结合使用了 Handler 与 Message。

```
public class MessageActivity extends AppCompatActivity implements OnClickListener {
    private TextView tv_message;
    private boolean bPlay = false;
    private int BEGIN = 0, SCROLL = 1, END = 2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_message);
        tv_message = (TextView) findViewById(R.id.tv_message);
        tv_message.setGravity(Gravity.LEFT|Gravity.BOTTOM);
        tv_message.setLines(8);
        tv_message.setMaxLines(8);
        tv_message.setMovementMethod(new ScrollingMovementMethod());
        findViewById(R.id.btn_start_message).setOnClickListener(this);
        findViewById(R.id.btn_stop_message).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_start_message) {
            if (bPlay != true) {
```

```
                    bPlay = true;
                    new PlayThread().start();
                }
            } else if (v.getId() == R.id.btn_stop_message) {
                bPlay = false;
            }
        }

        private String[] mNewsArray = { "中国航天员在天宫二号泡茶，羡煞歪果仁",
            "美国大选顺利闭幕，特朗普高票当选", "越南撤销日本获得订单的核电站计划",
            "上海建成卓越全球城市，新市镇轨交全覆盖", "土耳其老人怀抱受伤山羊错跑入医院急诊室" };
        private class PlayThread extends Thread {
            @Override
            public void run() {
                mHandler.sendEmptyMessage(BEGIN);
                while (bPlay == true) {
                    try {
                        sleep(2000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    Message message = Message.obtain();
                    message.what = SCROLL;
                    message.obj = mNewsArray[(int) (Math.random()*30%5)];
                    mHandler.sendMessage(message);
                }
                bPlay = true;
                try {
                    sleep(2000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                mHandler.sendEmptyMessage(END);
                bPlay = false;
            }
        }

        private Handler mHandler = new Handler() {
            @Override
            public void handleMessage(Message msg) {
                String desc = tv_message.getText().toString();
                if (msg.what == BEGIN) {
                    desc = String.format("%s\n%s %s", desc, DateUtil.getNowTime(), "下面开始播放新闻");
```

```
                } else if (msg.what == SCROLL) {
                    desc = String.format("%s\n%s %s", desc, DateUtil.getNowTime(), (String) msg.obj);
                } else if (msg.what == END) {
                    desc = String.format("%s\n%s %s", desc, DateUtil.getNowTime(), "新闻播放结束，谢
谢观看");
                }
                tv_message.setText(desc);
            };
        };
    }
```

新闻滚动的效果如图 10-1 与图 10-2 所示。其中，图 10-1 所示为正在播放新闻的界面，分线程每隔两秒添加一条新闻；图 10-2 所示为新闻播放结束时的界面，主线程收到分线程的 END 消息，在界面上提示用户“新闻播放结束，谢谢观看”。

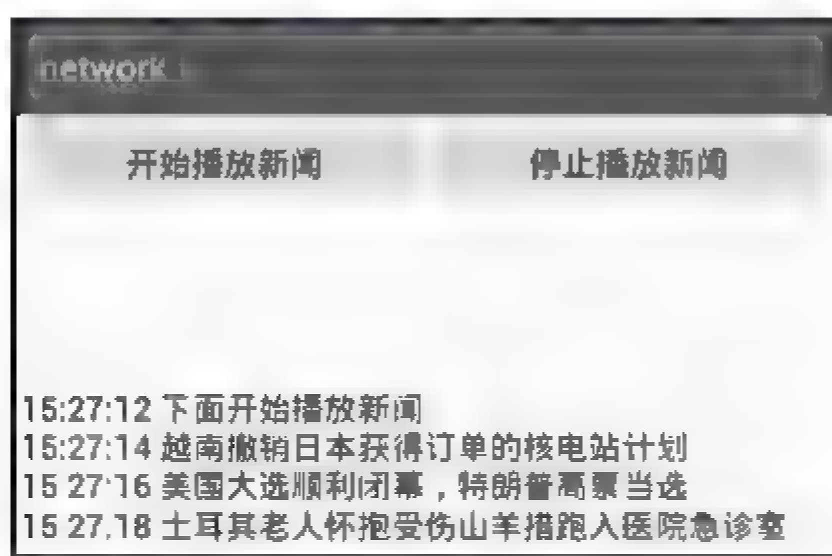


图 10-1　正在播放新闻的界面

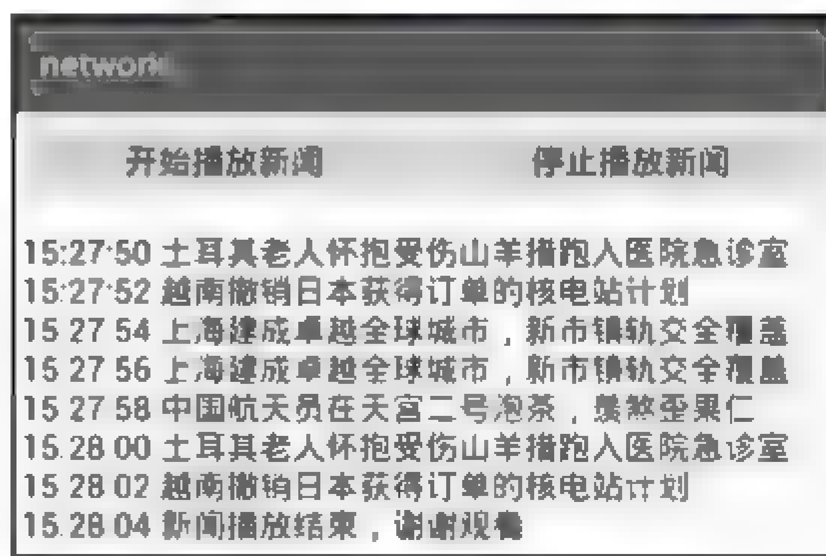


图 10-2　停止播放新闻的界面

10.1.2　进度对话框 ProgressDialog

有时，分线程在处理事务期间不允许用户继续操作界面控件，但是还想提示用户“页面正在加载，请耐心等待”之类信息，必要时还会告知用户当前的处理进度，这种情况就会用到进度对话框 ProgressDialog。分线程正在处理时，界面弹出进度对话框；分线程处理结束时，自动关闭进度对话框。这样既确保分线程不受干扰，又缓解了用户的焦急等待。

进度对话框继承自提醒对话框 AlertDialog，内部集成了进度条 ProgressBar，既拥有 AlertDialog 的所有方法，又实现了 ProgressBar 的公开 API。下面是进度对话框的常用方法。

- setTitle：设置对话框的标题文本。
- setMessage：设置对话框的消息内容。
- setIcon：设置对话框的图标。
- setProgress：设置当前进度的数值。
- setSecondaryProgress：设置当前第二进度的数值。
- setMax：设置进度条的最大进度数值。
- setProgressStyle：设置进度条的样式。取值 ProgressDialog.STYLE_SPINNER 表示转圈风格（默认值），取值 ProgressDialog.STYLE_HORIZONTAL 表示长条风格。
- show：显示对话框。需要在各属性设置完成后调用 show 方法。

- isShowing：判断对话框是否正在显示。
- dismiss：关闭对话框。
- 静态的 show 方法：简化的调用方法，一句代码就搞定进度对话框的设置与显示。可同时指定标题文字和消息内容，进度条样式为默认的转圈，示例代码如下：

```
ProgressDialog.show(this, "请稍候", "正在努力加载页面");
```

下面是使用进度对话框的代码片段：

```
private String[] descArray={"圆圈进度", "水平进度条"};
private int[] styleArray={ProgressDialog.STYLE_SPINNER, ProgressDialog.STYLE_HORIZONTAL};
private class StyleSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        if (mProgressDialog == null || mProgressDialog.isShowing() != true) {
            mStyleDesc = descArray[arg2];
            int style = styleArray[arg2];
            if (style == ProgressDialog.STYLE_SPINNER) {
                mProgressDialog = ProgressDialog.show(ProgressDialogActivity.this,
                        "请稍候", "正在努力加载页面");
                mHandler.postDelayed(mCloseDialog, 1500);
            } else {
                mProgressDialog = new ProgressDialog(ProgressDialogActivity.this);
                mProgressDialog.setTitle("请稍候");
                mProgressDialog.setMessage("正在努力加载页面");
                mProgressDialog.setMax(100);
                mProgressDialog.setProgressStyle(style);
                mProgressDialog.show();
                new RefreshThread().start();
            }
        }
    }

    public void onNothingSelected(AdapterView<?> arg0) {
    }
}

private Runnable mCloseDialog = new Runnable() {
    @Override
    public void run() {
        if (mProgressDialog.isShowing() == true) {
            mProgressDialog.dismiss();
            tv_result.setText(DateUtil.getNowTime()+" "+mStyleDesc+"加载完成");
        }
    }
};
```

```
private class RefreshThread extends Thread {
    @Override
    public void run() {
        for (int i=0; i<10; i++) {
            Message message = Message.obtain();
            message.what = 0;
            message.arg1 = i*10;
            mHandler.sendMessage(message);
            try {
                sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        mHandler.sendEmptyMessage(1);
    }
}

private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        if (msg.what == 0) {
            mProgressDialog.setProgress(msg.arg1);
        } else if (msg.what == 1) {
            post(mCloseDialog);
        }
    };
};
```

进度对话框的展示效果如图 10-3 与图 10-4 所示。其中，图 10-3 所示为转圈进度样式，对话框在 1.5 秒后自动关闭；图 10-4 所示为长条进度样式，每隔 0.5 秒进度数值增加 10，在 5 秒后关闭对话框。

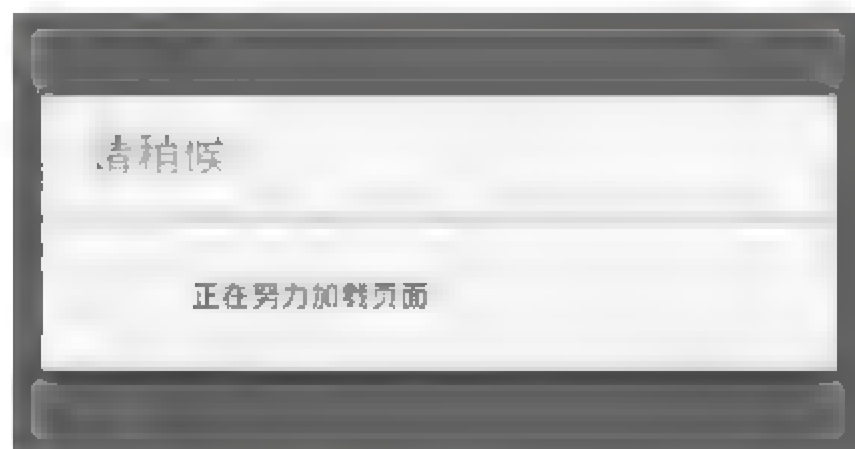


图 10-3　转圈样式的进度对话框



图 10-4　长条样式的进度对话框

当然，Android 默认的进度条并不好看，而且没有自带的进度文字提示，实际开发中往往要重新定制，使之符合用户的视觉习惯。主要的改造方向有两种：在长条进度中增加文字说明和在圆圈进度中增加文字说明。



1. 在长条样式中增加文字说明

修改长条样式的展示效果可通过定义层次图形并给 progressDrawable 属性赋值层次图形实现，具体方法参见第 6 章的“6.4.2 进度条 ProcessBar”。如果想在进度条中央显示进度文字，就得基于 ProgressBar 自定义一个进度条工具，主要思路是在 onDraw 方法中调用 canvas 的 drawText 方法往进度条上添加指定文本。

具体的代码实现不难，读者可尝试自行编码，也可参考本书的下载资源。文字进度条的显示效果如图 10-5 与图 10-6 所示。其中，图 10-5 是进度为 40%时的界面，图 10-6 是进度为 80%时的界面。



图 10-5　进度为 40%的进度条



图 10-6　进度为 80%的进度条

2. 在圆圈进度中增加文字说明

与长条进度相比，App 使用圆圈进度更加常见，可是 ProgressBar 的圆圈样式无法设定具体的进度值，若要采用圆圈进度，则必须完全摒弃 ProgressBar，从头实现自定义的圆圈进度工具。如果读者已仔细阅读本书前面的章节，相信你已经有了大概思路，就是利用第 6 章的自定义圆弧动画（参见“6.2.3 圆弧进度动画”）先画个背景圆环，再根据进度比例画个前景圆弧，最后在圆心处添加进度文本。

具体的实现代码不再赘述，读者可参照以上思路进行编码，也可参考本书的下载资源，自己动手实践看看。文字进度圈的显示效果如图 10-7 与图 10-8 所示。其中，图 10-7 是进度为 30%时的界面，图 10-8 是进度为 70%时的界面。

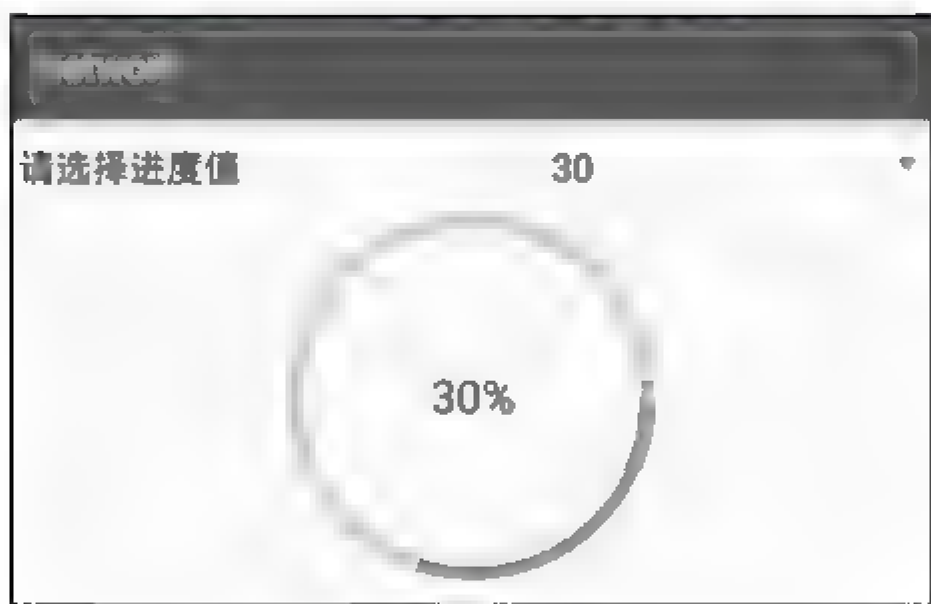


图 10-7　进度为 30%的进度圈

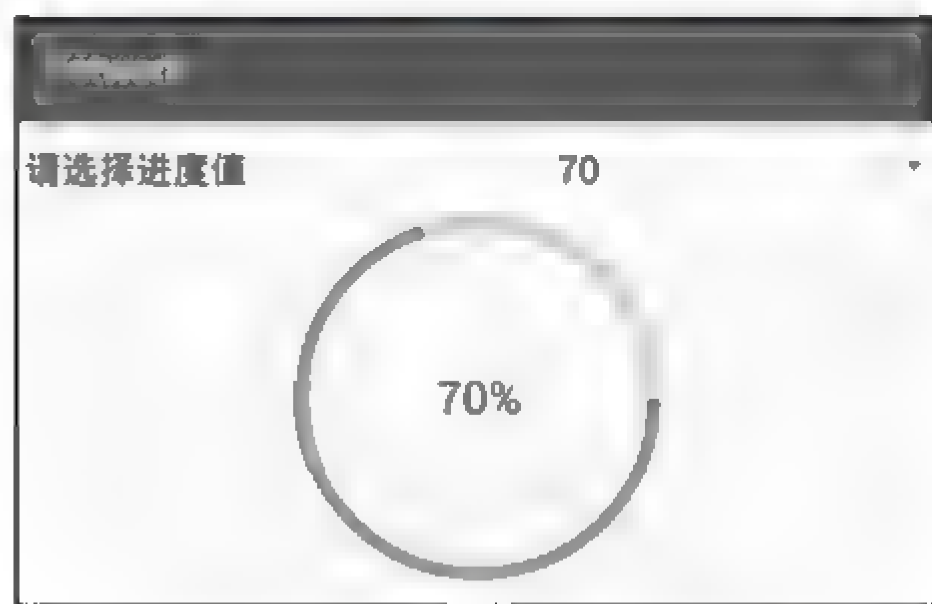


图 10-8　进度为 70%的进度圈

10.1.3　异步任务 AsyncTask

Thread+Handler 方式虽然能够实现多线程的通信处理，但是写起代码颇为麻烦，不但调用流程很烦琐，而且处理代码跟活动页面代码混在一起，非常不宜维护。基于以上问题，Android 提供了 AsyncTask 这个轻量级的异步任务工具，内部已经封装好 Thread+Handler 的线程通信

机制，开发者只需按部就班地编写业务代码，无须关心线程通信的复杂流程。AsyncTask 通常用于网络访问操作，包括 HTTP 接口调用、文件下载与上传等。

AsyncTask 是一个模板类（AsyncTask<Params, Progress, Result>），从它派生而来的新类需要指定模板的参数类型。下面来看模板参数说明。

- Params：任务启动时的输入参数，比如 HTTP 访问的 URL 地址、请求报文等。可设置为 String 类型或自定义的数据结构。
- Progress：任务执行过程中的进度。一般设置为 Integer 类型，表示当前处理进度。
- Result：任务执行完的结果参数，比如 HTTP 调用的执行结果、返回报文等。可设置为 String 类型或自定义的数据结构。

开发者自定义的任务类需要实现以下方法。

- onPreExecute：准备执行任务时触发。该方法在 doInBackground 方法执行之前调用。
- doInBackground：在后台执行的业务处理。网络请求等异步处理操作都放在该方法中，输入参数对应 execute 方法的输入参数，输出参数对应 onPostExecute 方法的输入参数。注意，该方法运行于分线程，不能操作界面，其他方法都能操作界面。
- onProgressUpdate：在 doInBackground 方法中调用 publishProgress 方法时触发。该方法通常用于在处理过程中刷新进度条。
- onPostExecute：任务执行完成时触发，方法内部可在页面上显示处理结果。该方法在 doInBackground 方法执行完毕后调用，输入参数对应 doInBackground 方法的输出参数。
- onCancelled :调用任务对象的 cancel 方法时触发。表示取消任务并返回。

另外，AsyncTask 有如下可直接调用的启停方法。

- execute：开始执行异步处理任务。
- executeOnExecutor：以指定的线程池模式执行任务。AsyncTask 内置的线程池模式有以下两个。
  - AsyncTask.THREAD_POOL_EXECUTOR：表示异步线程池（各任务间没有先后顺序，即有可能某任务在后面调用却先执行）。
  - AsyncTask.SERIAL_EXECUTOR：表示同步线程池（各任务按照代码调用的先后顺序依次排队等待执行），execute 方法默认使用 SERIAL_EXECUTOR。
- publishProgress：更新进度。该方法只能在 doInBackground 方法中调用，调用后会触发 onProgressUpdate 方法。
- get：获取处理结果。
- cancel：取消任务。该方法调用后，doInBackground 方法中的处理可能不会马上停止；若想立即停止处理，则可在 doInBackground 方法中加入 isCancelled 的判断。
- isCancelled：判断该任务是否取消。true 表示取消，false 表示未取消。
- getStatus：获取任务状态。任务状态的取值说明见表 10-1。

表10-1 任务状态的取值说明

AsyncTask.Status 类的任务状态	说明	所处时刻
PENDING	还未执行	onPreExecute 处理之前（正在等待）
RUNNING	正在执行	onPreExecute、doInBackground、onPostExecute 运行期间
FINISHED	执行完毕	onPostExecute 处理结束

下面是一个异步加载请求任务的代码：

```
public class ProgressAsyncTask extends AsyncTask<String, Integer, String> {
    private String mBook;
    public ProgressAsyncTask(String title) {
        super();
        mBook = title;
    }

    @Override
    protected String doInBackground(String... params) {
        int ratio = 0;
        for (; ratio <= 100; ratio += 5) {
            try {
                Thread.sleep(200);  // 睡眠 200 毫秒模拟网络通信处理
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            publishProgress(ratio);
        }
        return params[0];
    }

    @Override
    protected void onPreExecute() {
        mListener.onBegin(mBook);
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        mListener.onUpdate(mBook, values[0], 0);
    }

    @Override
    protected void onPostExecute(String result) {
        mListener.onFinish(result);
    }
```

```
    @Override
    protected void onCancelled(String result) {
        mListener.onCancel(result);
    }

    private OnProgressListener mListener;
    public void setOnProgressListener(OnProgressListener listener) {
        mListener = listener;
    }

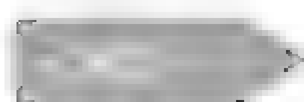
    public static interface OnProgressListener {
        public abstract void onFinish(String result);
        public abstract void onCancel(String result);
        public abstract void onUpdate(String request, int progress, int sub_progress);
        public abstract void onBegin(String request);
    }
}
```

在 Activity 中调用异步任务的完整代码如下：

```
public class AsyncTaskActivity extends AppCompatActivity implements OnProgressListener {
    private TextView tv_async;
    private ProgressBar pb_async;
    private ProgressDialog mDialog;
    public int mShowMode;
    public int BAR_HORIZONTAL = 1, DIALOG_CIRCLE = 2, DIALOG_HORIZONTAL = 3;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_async_task);
        tv_async = (TextView) findViewById(R.id.tv_async);
        pb_async = (ProgressBar) findViewById(R.id.pb_async);
        ArrayAdapter<String> styleAdapter = new ArrayAdapter<String>(this,
                R.layout.item_select, bookArray);
        Spinner sp_style = (Spinner) findViewById(R.id.sp_style);
        sp_style.setPrompt("请选择要加载的小说");
        sp_style.setAdapter(styleAdapter);
        sp_style.setOnItemSelectedListener(new StyleSelectedListener());
        sp_style.setSelection(0);
    }

    private String[] bookArray={"三国演义", "西游记", "红楼梦"};
    private int[] styleArray={BAR_HORIZONTAL, DIALOG_CIRCLE, DIALOG_HORIZONTAL};
```



```
class StyleSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        startTask(styleArray[arg2], bookArray[arg2]);
    }

    public void onNothingSelected(AdapterView<?> arg0) {
    }
}

private void startTask(int mode, String msg) {
    mShowMode = mode;
    ProgressAsyncTask asyncTask = new ProgressAsyncTask(msg);
    asyncTask.setOnProgressListener(this);
    asyncTask.execute(msg);
}

private void closeDialog() {
    if (mDialog != null && mDialog.isShowing() == true) {
        mDialog.dismiss();
    }
}

@Override
public void onFinish(String result) {
    String desc = String.format("您要阅读的《%s》已经加载完毕", result);
    tv_async.setText(desc);
    closeDialog();
}

@Override
public void onCancel(String result) {
    String desc = String.format("您要阅读的《%s》已经取消加载", result);
    tv_async.setText(desc);
    closeDialog();
}

@Override
public void onUpdate(String request, int progress, int sub_progress) {
    String desc = String.format("%s 当前加载进度为%d%%", request, progress);
    tv_async.setText(desc);
    if (mShowMode == BAR_HORIZONTAL) {
        pb_async.setProgress(progress);
        pb_async.setSecondaryProgress(sub_progress);
    } else if (mShowMode == DIALOG_HORIZONTAL) {
        mDialog.setProgress(progress);
        mDialog.setSecondaryProgress(sub_progress);
```

```
            }
        }

        @Override
        public void onBegin(String request) {
            tv_async.setText(request+"开始加载");
            if (mDialog == null || mDialog.isShowing() != true) {
                if (mShowMode == DIALOG_CIRCLE) {
                    mDialog = ProgressDialog.show(this, "稍等", request+"页面加载中……");
                } else if (mShowMode == DIALOG_HORIZONTAL) {
                    mDialog = new ProgressDialog(this);
                    mDialog.setTitle("稍等");
                    mDialog.setMessage(request+"页面加载中……");
                    mDialog.setIcon(R.drawable.ic_search);
                    mDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                    mDialog.show();
                }
            }
        }
    }
```

异步处理任务结合进度对话框的展示效果如图 10-9、图 10-10、图 10-11 所示。其中，图 10-9 所示为在页面上嵌入进度条的执行界面，图 10-10 所示为圆圈样式的进度对话框执行界面，图 10-11 所示为长条样式的进度对话框执行界面。

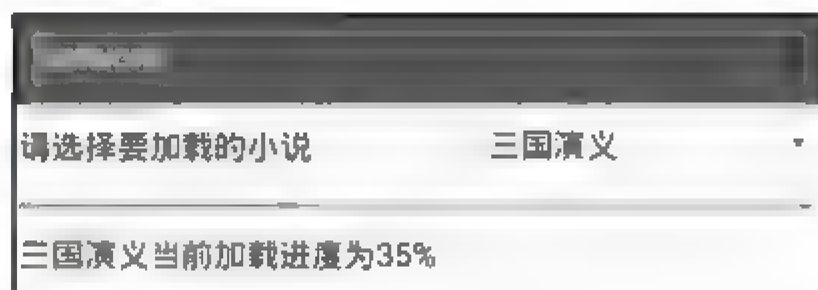


图 10-9　异步任务结合进度条的界面



图 10-10　异步任务结合转圈样式的界面

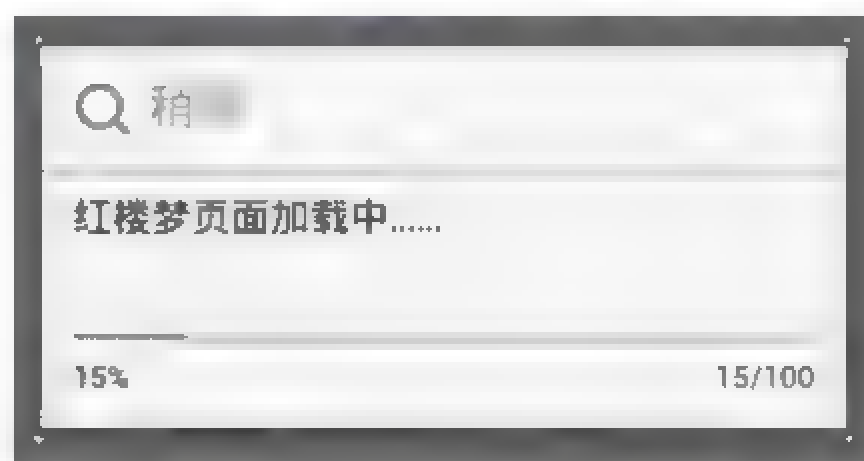


图 10-11　异步任务结合长条样式的界面

AsyncTask 在简单场合已经足够使用，如果要用于大量并发处理，就需要十分小心，因为 AsyncTask 的设计不甚完美，使用过程中要注意以下两点：

（1）AsyncTask 默认的线程池模式是 SERIAL_EXECUTOR，即按照先后顺序依次调用。假设有两个网络请求任务，第一个是文件下载，第二个是接口调用，那么接口调用任务会等待文件下载完毕后执行，而不是在调用时立刻执行。

（2）由于顺序模式存在排队等待的情况，因此 Android 提供了 executeOnExecutor 方法，允许开发者指定任务线程池。不过 AsyncTask 自带的 THREAD_POOL_EXECUTOR 也存在瓶颈，该线程池模式的最大线程个数是 CPU 个数的两倍再加 1（参见 AsyncTask 的源码

"MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1"）。如果用户手机采用了双核 CPU，那么 AsyncTask 的最大并发线程数为 2*2+1=5 个，此时若并发任务数超过 5 个，则后面进来的任务只能排队等待。如果用户手机采用的是四核 CPU，AsyncTask 的最大并发线程数就为 4*2+1=9 个，因此 CPU 个数越多，App 运行越流畅是有软件依据的。

10.1.4 异步服务 IntentService

服务 Service 虽然是在后台运行，但跟 Activity 一样都在主线程中，如果后台运行着的服务挂起，用户界面就会卡着不动，俗称死机。后台服务经常要做一些耗时操作，比如批量处理、文件导入、网络访问等，此时不应该影响用户在界面上的操作，而应该开启分线程执行耗时操作。可以通过 Thread+Handler 机制实现异步处理，也可以通过 Android 封装好的异步服务 IntentService 处理。

使用 IntentService 有两个好处，一个是免去复杂的消息通信流程；另一个是处理完成后无须手工停止服务，开发者可集中精力进行业务逻辑的编码。话虽如此，我们还是有必要了解一下 IntentService 的具体实现，入了这行一般都要干上许多年，晚学不如早学。前面提到，处理器对象位于主线程中，分线程通过 Handler 对象通知主线程，然后主线程执行 Handler 对象的 handleMessage 方法刷新界面。反过来也是允许的，即处理器对象位于分线程中，主线程通过 Handler 对象通知分线程，然后分线程执行 Handler 对象的 handleMessage 方法进行耗时处理。

具体请看 IntentService 的实现步骤。

01 创建异步服务时，初始化分线程的 Handler 对象，注意下面源码的 thread.getLooper 方法：

```
public void onCreate() {
    super.onCreate();
    HandlerThread thread = new HandlerThread("IntentService[" + mName + "]");
    thread.start();
    mServiceLooper = thread.getLooper();
    mServiceHandler = new ServiceHandler(mServiceLooper);
}
```

02 异步服务开始运行时，通过 Handler 对象将请求数据送给分线程，源码如下：

```
public void onStart(Intent intent, int startId) {
    Message msg = mServiceHandler.obtainMessage();
    msg.arg1 = startId;
    msg.obj = intent;
    mServiceHandler.sendMessage(msg);
}
```

03 分线程在 Handler 对象的 handleMessage 方法中，先通过 onHandleIntent 方法执行具体的事务处理，再调用 stopSelf 结束指定标识的服务。源码如下：

```
private final class ServiceHandler extends Handler {
    public ServiceHandler(Looper looper) {
        super(looper);
```

```
            }

            @Override
            public void handleMessage(Message msg) {
                onHandleIntent((Intent)msg.obj);
                stopSelf(msg.arg1);
            }
        }
```

了解 IntentService 的实现思想后，使用过程中需要注意以下 4 点：

（1）增加一个构造方法，并分配内部线程的唯一名称。

（2）onStartCommand 方法要调用父类的 onStartCommand，因为父类方法会向分线程传递消息。

（3）耗时处理的业务代码要写在 onHandleIntent 方法中，不可写在 onStartCommand 方法中。因为 onHandleIntent 方法位于分线程，而 onStartCommand 方法位于主线程。

（4）IntentService 实现了 onStart 方法，却未实现 onBind 方法，意味着异步服务只能用普通方式启停，不能用绑定方式启停。

下面是使用异步服务的代码：

```
public class AsyncService extends IntentService {
    private static final String TAG = "AsyncService";
    public AsyncService() {
        super("com.example.network.service.AsyncService");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startid) {
        Log.i(TAG, "onStartCommand");
        //试试在 onStartCommand 里调用 Thread.sleep 方法，页面按钮是不是无法点击了
        return super.onStartCommand(intent, flags, startid);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        Log.d(TAG, "begin onHandleIntent");
        //在 onHandleIntent 中执行耗时任务，不会影响页面的处理
        try {
            Thread.sleep(30*1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        Log.d(TAG, "end onHandleIntent");
    }
}
```

异步服务的演示效果如图 10-12 所示，即使异步服务在 onHandleIntent 方法中睡眠 30 秒，也丝毫不影响用户在页面上的点击操作。读者可以尝试在 onStartCommand 方法中睡眠 30 秒，看看能否在页面上正常点击按钮。

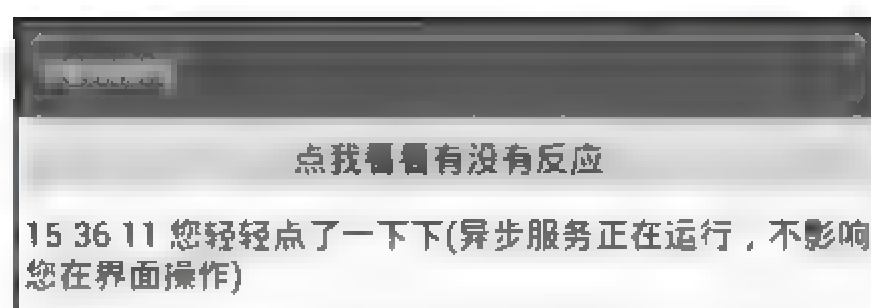


图 10-12 异步服务的演示效果图

10.2 HTTP 接口访问

本节介绍 HTTP 接口访问的相关技术与具体使用，首先说明如何利用连接管理器 ConnectivityManager 检测网络连接的状态；然后阐述 App 用于接口调用的移动数据格式 JSON 的构建与解析；接着举例说明通过 HttpURLConnection 实现基本的接口调用，包括 GET 和 POST 两种常见的调用方式，并给出阶段性实战项目“根据经纬度获取地址信息”的实现过程；最后讲述利用 HttpURLConnection 从网络获取小图片的方法。

10.2.1 网络连接检查

谈到网络通信，首先要检查当前是否处于上网状态，然后进行网络访问操作。如果当前网络连接不可用，那么无须执行网络访问，直接提示用户“请开启网络连接”就好了。要检测网络连接，Android 会要求 App 具备上网权限，所以首先打开 AndroidManifest.xml，加上下面几行网络权限配置：

```
<!-- 互联网 -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- 查看网络状态 -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

添加网络权限配置后，可利用连接管理器 ConnectivityManager 检测网络连接，该工具的对象从系统服务 Context.CONNECTIVITY_SERVICE 中获取。调用连接管理器对象的 getActiveNetworkInfo 方法，返回一个 NetworkInfo 实例，通过该实例可获取详细的网络连接信息。下面是 NetworkInfo 的常用方法。

- getType：获取网络类型。网络类型的取值说明见表 10-2。

表10-2 网络类型的取值说明

ConnectivityManager 类的网络类型	说明
TYPE_WIFI	wifi
TYPE_MOBILE	数据连接
TYPE_WIMAX	wimax

（续表）

ConnectivityManager 类的网络类型	说明
TYPE_ETHERNET	以太网
TYPE_BLUETOOTH	蓝牙
TYPE_VPN	vpn

- getState：获取网络状态。网络状态的取值说明见表 10-3。

表10-3　网络状态的取值说明

NetworkInfo.State 的网络状态	说明
CONNECTING	正在连接
CONNECTED	已连接
SUSPENDED	挂起
DISCONNECTING	正在断开
DISCONNECTED	已断开
UNKNOWN	未知

- getSubtype：获取网络子类型。当网络类型为数据连接时，子类型为 2G/3G/4G 的细分类型，如 CDMA、EVDO、HSDPA、LTE 等。网络子类型的取值说明见表 10-4。

表10-4　网络子类型的取值说明

取值	TelephonyManager 类的网络子类型	制式分类
1	NETWORK_TYPE_GPRS	2G
2	NETWORK_TYPE_EDGE	2G
3	NETWORK_TYPE_UMTS	3G
4	NETWORK_TYPE_CDMA	2G
5	NETWORK_TYPE_EVDO_0	3G
6	NETWORK_TYPE_EVDO_A	3G
7	NETWORK_TYPE_1xRTT	2G
8	NETWORK_TYPE_HSDPA	3G
9	NETWORK_TYPE_HSUPA	3G
10	NETWORK_TYPE_HSPA	3G
11	NETWORK_TYPE_IDEN	2G
12	NETWORK_TYPE_EVDO_B	3G
13	NETWORK_TYPE_LTE	4G
14	NETWORK_TYPE_EHRPD	3G
15	NETWORK_TYPE_HSPAP	3G
16	NETWORK_TYPE_GSM	2G
17	NETWORK_TYPE_TD_SCDMA	3G
18	NETWORK_TYPE_IWLAN	4G



网络连接的检测结果如图 10-13 和图 10-14 所示。其中，图 10-13 表示当前处于 WIFI 环境，图 10-14 表示当前使用 4G 类型的数据连接上网。



图 10-13　连接 WIFI 的检测结果图

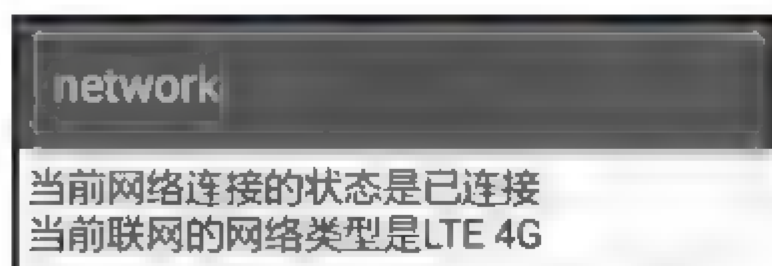


图 10-14　数据连接的检测结果图

10.2.2　移动数据格式 JSON

网络通信的交互数据格式有两大类，分别是 JSON 和 XML，前者短小精悍，后者表现力丰富。对于 App 来说，基本采用 JSON 格式与服务器通信。原因很多，一个是手机流量很贵，表达同样的信息，JSON 串比 XML 串短很多，在节省流量方面占了上风；另一个是 JSON 串解析得更快，也更省电，XML 不但慢而且耗电。于是，JSON 格式成了移动端事实上的网络数据格式标准。

Android 自带 JSON 解析工具，提供对 JSONObject（JSON 对象）和 JSONArray（JSON 数组）的解析处理。

1. JSONObject

下面来看 JSONObject 的常用方法。

- JSONObject 构造函数：从指定字符串构造一个 JSONObject 对象。
- getJSONObject：获取指定名称的 JSONObject 对象。
- getString：获取指定名称的字符串。
- getInt：获取指定名称的整型数。
- getDouble：获取指定名称的双精度数。
- getBoolean：获取指定名称的布尔数。
- getJSONArray：获取指定名称的 JSONArray 数组对象。
- put：添加一个 JSONObject 对象。
- toString：把当前的 JSONObject 对象输出为一个 JSON 字符串。

2. JSONArray

下面来看 JSONArray 的常用方法。

- length：获取 JSONArray 数组的长度。
- getJSONObject：获取 JSONArray 数组在指定位置的 JSONObject 对象。
- put：往 JSONArray 数组中添加一个 JSONObject 对象。

下面是使用 JSON 串的代码片段，包括如何构造 JSON 串和如何解析 JSON 串：

```
// 构造 JSON 串
private String getJsonStr() {
```



```
        String str = "";
        JSONObject obj = new JSONObject();
        try {
            obj.put("name", "address");
            JSONArray array = new JSONArray();
            for (int i = 0; i < 3; i++) {
                JSONObject item = new JSONObject();
                item.put("item", "第" + (i + 1) + "个元素");
                array.put(item);
            }
            obj.put("list", array);
            obj.put("count", array.length());
            obj.put("desc", "这是测试串");
            str = obj.toString();
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return str;
    }

    // 解析 JSON 串
    private String parserJson(String jsonStr) {
        String result = "";
        try {
            JSONObject obj = new JSONObject(jsonStr);
            String name = obj.getString("name");
            String desc = obj.getString("desc");
            int count = obj.getInt("count");
            result = String.format("%sname=%s\n", result, name);
            result = String.format("%sdesc=%s\n", result, desc);
            result = String.format("%scount=%d\n", result, count);
            JSONArray listArray = obj.getJSONArray("list");
            for (int i=0; i<listArray.length(); i++) {
              JSONObject list_item = listArray.getJSONObject(i);
              String item = list_item.getString("item");
                result = String.format("%s\titem=%s\n", result, item);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
        return result;
    }
```

示例代码对应的效果如图 10-15 和图 10-16 所示。其中，图 10-15 所示为构造 JSON 串的结果界面，图 10-16 所示为解析 JSON 串的结果界面。

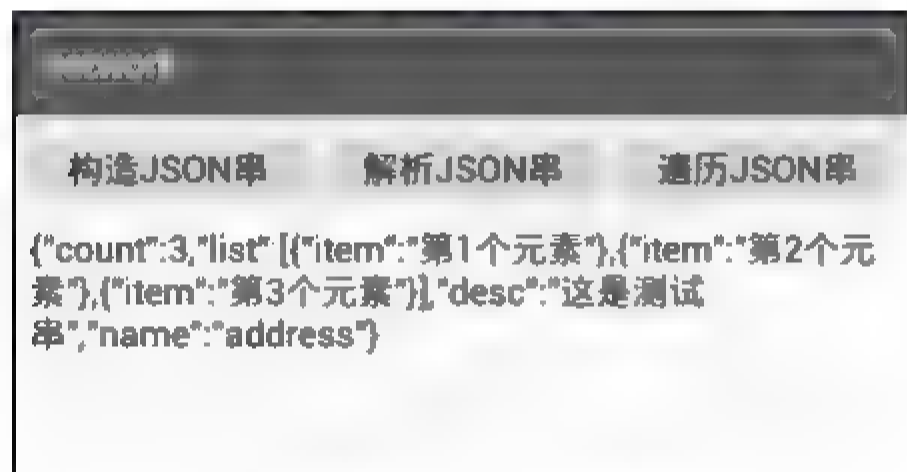


图 10-15　构造 JSON 串的结果图

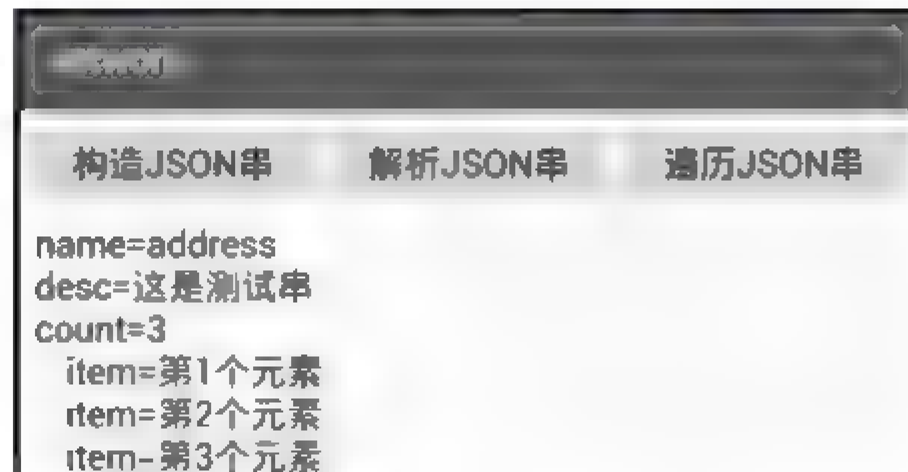


图 10-16　解析 JSON 串的结果图

10.2.3　HTTP 接口调用

HTTP 接口调用的代码标准有两个，分别是 HttpURLConnection 与 HttpClient。就像 JSON 与 XML 的区别一样，移动端的代码标准基本采用更轻量级的 HttpURLConnection。只使用 HttpURLConnection 就能玩转几乎所有 HTTP 访问，当然复杂的功能（如分段传输、上传等）得自己写代码细节。

HttpURLConnection 对象从 URL 对象的 openConnection 方法获得。下面来看该对象的常用方法。

- setRequestMethod：设置请求类型。GET 表示 get 请求，POST 表示 post 请求。
- setConnectTimeout：设置连接的超时时间。
- setReadTimeout：设置读取的超时时间。
- setRequestProperty：设置请求包头的属性信息。
- setDoOutput：设置是否允许发送数据。如果用到 getOutputStream 方法，setDoOutput 就必须设置为 true。因为 POST 方式肯定会发送数据，所以 POST 调用时必须设置该方法。
- getOutputStream：获取 HTTP 输出流。调用该函数返回一个 OutputStream 对象，接着依次调用该对象的 write 和 flush 方法写入要发送的数据。
- connect：建立 HTTP 连接。该方法在 getOutputStream 后调用，在 getInputStream 前调用。
- setDoInput：设置是否允许接收数据。如果用到 getInputStream 方法，setDoInput 就必须设置为 true（其实也不必手动设置，因为默认就是 true）。
- getInputStream：获取 HTTP 输入流。调用该函数返回一个 InputStream 对象，接着调用该对象的 read 方法读出接收的数据。
- getResponseCode：获取 HTTP 返回码。
- getHeaderField：获取应答数据包头的指定属性值。
- getHeaderFields：获取应答数据包头的所有属性列表。
- disconnect：断开 HTTP 连接。

HTTP 接口调用主要有 GET 和 POST 两种方式，GET 方式只是简单的数据获取操作，类似于数据库的查询操作；POST 方式有提交具体的表单信息，类似于数据库的增、删、改操作。两种接口调用都有固定的代码模板，直接套用即可。下面是 HTTP 接口调用的代码：

```
//设置默认的连接属性信息
private static void setConnHeader(HttpURLConnection conn, String method, HttpReqData req_data)
            throws ProtocolException {
        conn.setRequestMethod(method);
        conn.setConnectTimeout(5000);
        conn.setReadTimeout(10000);
        conn.setRequestProperty("Accept", "*/*");
        conn.setRequestProperty("Accept-Language", "zh-CN");
        conn.setRequestProperty("Accept-Encoding", "gzip, deflate");
        if (req_data.content_type.equals("") != true) {
            conn.setRequestProperty("Content-Type", req_data.content_type);
        }
}

//get 文本数据
public static HttpRespData getData(HttpReqData req_data) {
        HttpRespData resp_data = new HttpRespData();
        try {
            URL url = new URL(req_data.url);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            setConnHeader(conn, "GET", req_data);
            conn.connect();
            resp_data.content = StreamTool.getUnzipStream(conn.getInputStream(),
                    conn.getHeaderField("Content-Encoding"), req_data.charset);
            resp_data.cookie = conn.getHeaderField("Set-Cookie");
            conn.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
            resp_data.err_msg = e.getMessage();
        }
        return resp_data;
}

//post 的内容放在 url 中
public static HttpRespData postUrl(HttpReqData req_data) {
        HttpRespData resp_data = new HttpRespData();
        String s_url = req_data.url;
        if (req_data.params != null) {
            s_url += "?" + req_data.params.toString();
        }
        Log.d(TAG, "s_url="+s_url);
        try {
            URL url = new URL(s_url);
```

```
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            setConnHeader(conn, "POST", req_data);
            conn.setDoOutput(true);
            conn.connect();
            resp_data.content = StreamTool.getUnzipStream(conn.getInputStream(),
                    conn.getHeaderField("Content-Encoding"), req_data.charset);
            resp_data.cookie = conn.getHeaderField("Set-Cookie");
            conn.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
            resp_data.err_msg = e.getMessage();
        }
        return resp_data;
    }

    //post 的内容放在输出流中
    public static HttpRespData postData(HttpReqData req_data) {
        req_data.content_type = "application/x-www-form-urlencoded";
        HttpRespData resp_data = new HttpRespData();
        String s_url = req_data.url;
        Log.d(TAG, "s_url="+s_url+", params="+req_data.params.toString());
        try {
            URL url = new URL(s_url);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            setConnHeader(conn, "POST", req_data);
            conn.setDoOutput(true);
            conn.setDoInput(true);
            conn.connect();
            PrintWriter out = new PrintWriter(conn.getOutputStream());
            out.print(req_data.params.toString());
            out.flush();
            resp_data.content = StreamTool.getUnzipStream(conn.getInputStream(),
                    conn.getHeaderField("Content-Encoding"), req_data.charset);
            resp_data.cookie = getRespCookie(conn, req_data);
            conn.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
            resp_data.err_msg = e.getMessage();
        }
        return resp_data;
    }
}
```

正所谓好事多磨，HTTP 访问除了套用调用模板外，还要处理好几种特殊情况，否则就不会正常工作。常见的特殊情况有两种：URL 串中对汉字的转义处理和返回内容为压缩数据时的解压处理。

1. URL 串中对汉字的转义处理

使用 GET 方式传递请求数据，参数放在 URL 中直接传送过去。如果参数值有汉字，就进行 UTF8 编码转义处理，比如“你”要转为“%E4%BD%A0”。同理，对于服务器返回的 UTF8 编码也要进行反转义，比如“%E4%BD%A0”要转为“你”。具体的转义代码参见本书下载资源的 URLtoUTF8.java。

2. 返回内容为压缩数据时的解压处理

HTTP 请求的包头带有 Accept-Encoding：gzip,deflate，表示客户端支持 gzip 压缩。服务器可能返回 gzip 压缩的应答数据，此时应答包头中会有 Content-Encoding：gzip。此时压缩数据必须先解压才能正常读取，未解压只会读到一堆乱码。输入流的 gzip 解压使用 GZIPInputStream 工具类，具体的解压代码参见本书下载资源的 StreamTool.java。

下面用一个阶段性的实战小项目练练手。第 9 章在介绍定位功能时使用定位管理器获取手机的位置信息，包括经度、纬度、高度等，不过用户关心的是具体的地址描述，而不是看不懂的经纬度。现在我们利用 Google Map 的开放 API，通过 HTTP 调用传入经纬度的数值，然后对方返回一个 JSON 格式的地址信息字符串，通过解析 JSON 串就能得到具体的地址。

因为网络访问不能在主线程中进行，所以要结合 AsyncTask 与 HttpURLConnection 实现地址的异步获取。获取地址信息的任务代码示例如下：

```
public class GetAddressTask extends AsyncTask<Location, Void, String> {
    private final static String TAG = "GetAddressTask";
    private static String mAddressUrl = "http://maps.google.cn/maps/api/geocode/json?latlng={0},
{1}&sensor=true&language=zh-CN";
    public GetAddressTask() {
        super();
    }

    @Override
    protected String doInBackground(Location... params) {
        Location location = params[0];
        String url = MessageFormat.format(mAddressUrl, location.getLatitude(), location.getLongitude());
        HttpReqData req_data = new HttpReqData(url);
        HttpRespData resp_data = HttpRequestUtil.getData(req_data);
        Log.d(TAG, "return json = " + resp_data.content);

        String address = "未知";
        if (resp_data.err_msg.length() <= 0) {
            try {
```



```
                    JSONObject obj = new JSONObject(resp_data.content);
                    JSONArray resultArray = obj.getJSONArray("results");
                    if (resultArray.length() > 0) {
                        JSONObject resultObj = resultArray.getJSONObject(0);
                        address = resultObj.getString("formatted_address");
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
            Log.d(TAG, "address = " + address);
            return address;
        }

        @Override
        protected void onPostExecute(String address) {
            mListener.onFindAddress(address);
        }

        private OnAddressListener mListener;
        public void setOnAddressListener(OnAddressListener listener) {
            mListener = listener;
        }

        public static interface OnAddressListener {
            public abstract void onFindAddress(String address);
        }
}
```

接着在原来的Activity代码中启动该任务，并实现OnAddressListener接口的onFindAddress方法，即可在页面上添加详细的地址信息。启动任务的代码如下：

```
                GetAddressTask addressTask = new GetAddressTask();
                addressTask.setOnAddressListener(this);
                addressTask.execute(location);
```

定位并获取地址信息的效果如图 10-17 所示。此时除了原来的经纬度数据外，还多了一个文字表达的详细地址，从省、市、区一直到具体的街道和门牌号。如此一来，定位功能的实用性就大大增强了。

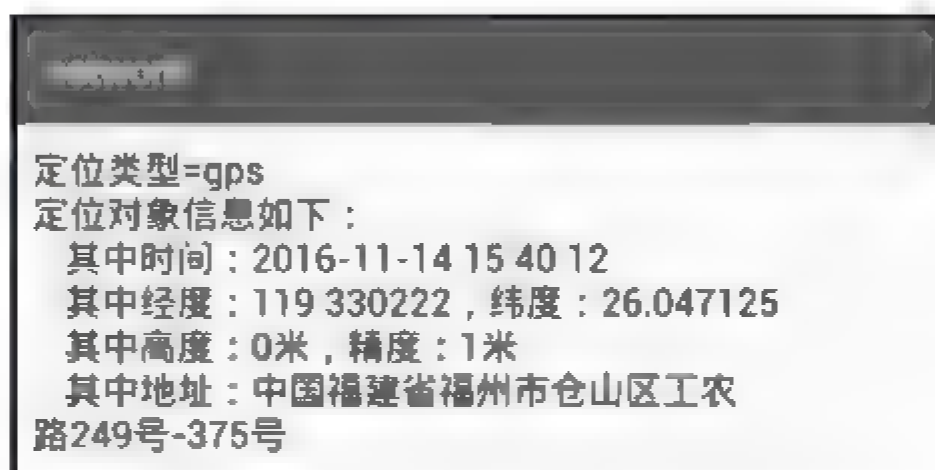


图 10-17 通过HTTP调用获得地址信息的效果图

10.2.4 HTTP 图片获取

除了 HTTP 接口调用外，HttpURLConnection 还可用于获取网络小图片，比如验证码图片、头像图标等，这些小图不大，一般也无须缓存，可直接从网络上获取最新的图片。

下面是使用 HttpURLConnection 获取图片的代码：

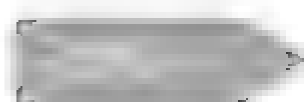
```
//get 图片数据
public static HttpRespData getImage(HttpReqData req_data) {
        HttpRespData resp_data = new HttpRespData();
        try {
                URL url = new URL(req_data.url);
                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                setConnHeader(conn, "GET", req_data);
                conn.connect();
                InputStream is = conn.getInputStream();
                resp_data.bitmap = BitmapFactory.decodeStream(is);
                resp_data.cookie = conn.getHeaderField("Set-Cookie");
                conn.disconnect();
        } catch (Exception e) {
                e.printStackTrace();
                resp_data.err_msg = e.getMessage();
        }
        return resp_data;
}
```

在活动页面与 HTTP 图片获取之间还需一个基于 AsyncTask 的图片获取任务做桥梁。下面是获取图片验证码的任务代码：

```
public class GetImageCodeTask extends AsyncTask<Void, Void, String> {
        private final static String TAG = "GetImageCodeTask";
        private Context mContext;
        private String mImageCodeUrl = "http://220.160.54.47:82/JSPORTLET/radomImage?x=";

        public GetImageCodeTask(Context context) {
                super();
                mContext = context;
        }

        @Override
        protected String doInBackground(Void... params) {
                String url = mImageCodeUrl + DateUtil.getNowDateTime(null);
                Log.d(TAG, "image url="+url);
                HttpReqData req_data = new HttpReqData(url);
                HttpRespData resp_data = HttpRequestUtil.getImage(req_data);
                String path = BitmapUtil.getCachePath(mContext) + DateUtil.getNowDateTime(null) + ".jpg";
```



```
            BitmapUtil.saveBitmap(path, resp_data.bitmap, "jpg", 80);
            Log.d(TAG, "image path="+path);
            return path;
        }

        @Override
        protected void onPostExecute(String path) {
            mListener.onGetCode(path);
        }

        private OnImageCodeListener mListener;
        public void setOnImageCodeListener(OnImageCodeListener listener) {
            mListener = listener;
        }

        public static interface OnImageCodeListener {
            public abstract void onGetCode(String path);
        }
    }
```

下面是在页面代码中调用验证码获取任务的代码片段，首先指定 task 任务，然后实现 onGetCode 方法显示验证码图片：

```
        private void getImageCode() {
            if (bRunning != true) {
                bRunning = true;
                GetImageCodeTask codeTask = new GetImageCodeTask(this);
                codeTask.setOnImageCodeListener(this);
                codeTask.execute();
            }
        }

        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.iv_image_code) {
                getImageCode();
            }
        }

        @Override
        public void onGetCode(String path) {
            Uri uri = Uri.parse(path);
            iv_image_code.setImageURI(uri);
            bRunning = false;
        }
```

从网络上获取并显示验证码图片的效果如图 10-18 和图 10-19 所示。其中，图 10-18 所示为页面的初始界面，点击图片后会重新加载验证码；图 10-19 所示为验证码图片刷新后的界面。



图 10-18　获取验证码图片的初始页面

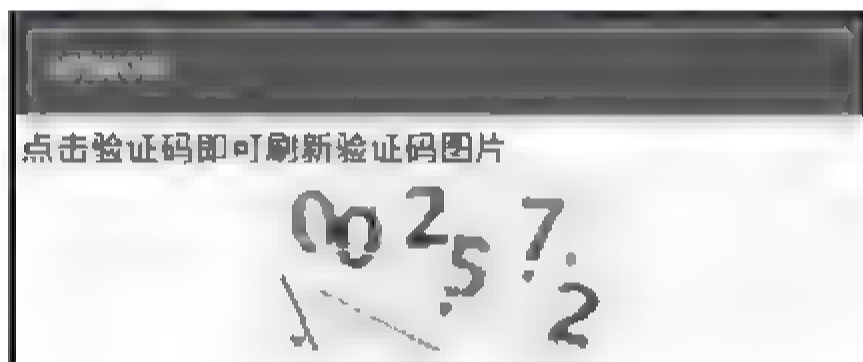


图 10-19　验证码图片刷新后的页面

10.3　上传和下载

本节介绍 App 与服务器之间上传文件和下载文件的实现与管理，首先对下载管理器 DownloadManager 进行详细说明，包括文件下载的 3 个步骤、3 种下载事件以及下载进度的两种查看方式（通知栏查看和游标轮询）；然后阐述基于 Fragment 技术的文件对话框实现，包括文件保存对话框和文件打开对话框两种形式；最后介绍通过 HttpURLConnection 的 POST 方式如何实现文件的上传操作，以及上传服务器的简单搭建过程。

10.3.1　下载管理器 DownloadManager

10.2 节提到使用 HttpURLConnection 可以获取小图片，不过这么做有诸多限制，比如：

（1）无法断点续传，一旦中途失败，只能从头开始获取。
（2）只能获取图片，不能获取其他文件。
（3）不是真正意义上的下载操作，没法设置下载参数。

所以，10.2 节的做法只能用于获取小图，如果要下载大图或下载其他格式的文件就要另想办法。因为下载功能比较常用且业务功能相对统一，所以 Android 从 2.3（API9）开始提供了专门的下载工具—— DownloadManager 统一管理下载操作。

下载管理器 DownloadManager 的对象从系统服务 Context.DOWNLOAD_SERVICE 中获取，具体使用过程分为 3 步：构建下载请求、进行下载操作和查询下载进度。

1. 构建下载请求

要想使用下载功能，首先得构建一个下载请求，说明从哪里下载、下载参数是什么、下载的文件保存到哪里等。这个下载请求就是 DownloadManager 的内部类 Request。下面来看该类的常用方法说明。

- 构造函数：指定从哪个网络地址下载文件。
- setAllowedNetworkTypes：指定允许下载的网络类型。允许网络类型的取值说明见表 10-5。若同时允许多种网络类型，则可使用竖线“|”把多种网络类型拼接起来。

表10-5　允许网络类型的取值说明

DownloadManager.Request 类的允许网络类型	说明
NETWORK_WIFI	WIFI 网络
NETWORK_MOBILE	数据连接网络
NETWORK_BLUETOOTH	蓝牙网络

- setDestinationInExternalFilesDir：设置下载文件在本地的保存路径。第二个参数为目录类型，取值说明见第 4 章的表 4-2；第三个参数为不带斜杆的文件名；另外，如果指定目录已存在同名文件，系统就会将新下载的文件重命名，即在文件名末尾添加“-1”“-2”之类的序号。
- addRequestHeader：给 HTTP 请求添加头部参数。
- setMimeType：设置下载文件的媒体类型。一般无须设置，默认是服务器返回的媒体类型。
- setTitle：设置通知栏上的消息标题。如果不设置，默认标题就是下载的文件名。
- setDescription：设置通知栏上的消息描述。如果不设置，就默认显示系统估算的下载剩余时间。
- setVisibleInDownloadsUi：设置是否显示在系统的下载页面上。
- setNotificationVisibility：设置通知栏的下载任务可见类型。可见类型的取值说明见表 10-6。

表10-6　通知可见类型的取值说明

DownloadManager.Request 类的通知可见类型	说明
VISIBILITY_HIDDEN	隐藏
VISIBILITY_VISIBLE	下载时可见（下载完成后消失）
VISIBILITY_VISIBLE_NOTIFY_COMPLETED	下载进行时与完成后都可见
VISIBILITY_VISIBLE_NOTIFY_ONLY_COMPLETION	只有下载完成后可见

2. 进行下载操作

构建完下载请求才能进行下载的相关操作。下面是 DownloadManager 的常用方法。

- enqueue：将下载请求加入任务队列中，排队等待下载。该方法返回本次下载任务的编号。
- remove：取消指定编号的下载任务。
- restartDownload：重新开始指定编号的下载任务。
- openDownloadedFile：打开下载完成的文件。
- getMimeTypeForDownloadedFile：获取下载完成文件的媒体类型。
- query：根据查询请求获取符合条件的结果集游标。

3. 查询下载进度

虽然下载进度可在通知栏上查看，但是如果 App 自身也想了解当前的下载进度，就要调用下载管理器的 query 方法。该方法的输入参数是一个 Query 对象，返回结果集的 Cursor 游标，这里的 Cursor 用法与 SQLite 里的 Cursor 一样，具体可参考第 4 章的“4.2 数据库 SQLite”。



下面是 Query 类的常用方法说明。

- setFilterById：根据编号过滤下载任务。
- setFilterByStatus：根据状态过滤下载任务。
- setOnlyIncludeVisibleInDownloadsUi：是否只包含在系统下载页面上的可见任务。
- orderBy：结果集按照指定字段排序。

设置完查询请求，即可调用 DownloadManager 对象的 query 方法，获得结果集的游标对象。该游标中包含下载任务的完整字段信息，主要下载字段的取值说明见表 10-7。

表10-7　下载字段的取值说明

DownloadManager 类的下载字段	说明
COLUMN_LOCAL_FILENAME	下载文件的本地保存路径
COLUMN_MEDIA_TYPE	下载文件的媒体类型
COLUMN_TOTAL_SIZE_BYTES	下载文件的总大小
COLUMN_BYTES_DOWNLOADED_SO_FAR	已下载的文件大小
COLUMN_STATUS	下载状态。下载状态的取值说明见表 10-8

表10-8　下载状态的取值说明

DownloadManager 类的下载状态	说明
STATUS_PENDING	挂起，即正在等待
STATUS_RUNNING	运行中
STATUS_PAUSED	暂停
STATUS_SUCCESSFUL	成功
STATUS_FAILED	失败

另外，系统的下载服务还提供 3 种下载事件，开发者可通过监听对应的广播消息进行相应的处理。3 种下载事件说明如下：

1. 下载完成事件

在下载完成时，系统会发出名为 DownloadManager.ACTION_DOWNLOAD_COMPLETE（值为字符串 android.intent.action.DOWNLOAD_COMPLETE）的广播，因此可注册一个该广播的接收器，用来判断当前任务是否已下载完毕，并进行后续的业务处理。

2. 下载进行时的通知栏点击事件

在下载过程中，只要用户点击通知栏上的下载任务，系统就会发出行为名称是 DownloadManager.ACTION_NOTIFICATION_CLICKED（值为字符串 android.intent.action.DOWNLOAD_NOTIFICATION_CLICKED）的广播，可注册该广播的接收器进行相关处理，比如跳转到该任务的下载进度页面等。



3. 下载完成后的通知栏点击事件

在不同时刻点击通知栏上的下载任务会触发不同的事件。下载未完成时点击触发的是系统广播 DownloadManager.ACTION_NOTIFICATION_CLICKED。下载完成后点击触发的是系统的 Intent.ACTION_VIEW（浏览行为）。对于浏览行为，系统会根据媒体类型自动寻找对应App打开。因此，如果开发者要控制此时的点击行为，可以调用 Request 对象的 setMimeType 方法设置媒体类型，这样 Android 就会按照这个类型打开相应的 App。

下面是利用 DownloadManager 下载 APK 安装包的代码片段，下载进度显示在通知栏上：

```
    private class ApkUrlSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            sp_apk_url.setEnabled(false);
            Uri uri = Uri.parse(apkUrlArray[arg2]);
            Request down = new Request(uri);
            down.setTitle(apkDescArray[arg2]+"下载信息");
            down.setDescription(apkDescArray[arg2]+"安装包正在下载");
            down.setAllowedNetworkTypes(Request.NETWORK_MOBILE | Request.NETWORK_WIFI);
            down.setNotificationVisibility(Request.VISIBILITY_VISIBLE_NOTIFY_COMPLETED);
            down.setVisibleInDownloadsUi(true);
            down.setDestinationInExternalFilesDir(DownloadApkActivity.this,
                    Environment.DIRECTORY_DOWNLOADS, arg2 + ".apk");
            mDownloadId = mDownloadManager.enqueue(down);
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }

    // 接收下载完成事件
    public static class DownloadCompleteReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            if (intent.getAction().equals(DownloadManager.ACTION_DOWNLOAD_COMPLETE)
                    && tv_apk_result != null) {
                long downId = intent.getLongExtra(DownloadManager.EXTRA_DOWNLOAD_ID, -1);
                Log.d(TAG, " download complete! id ： " + downId + ", mDownloadId=" +
mDownloadId);
                tv_apk_result.setVisibility(View.VISIBLE);
                tv_apk_result.setText(DateUtil.getNowDateTime(null) + " 编号"
                        + downId + "的下载任务已完成");
                sp_apk_url.setEnabled(true);
            }
        }
```

```
    }

    // 接收下载通知栏的点击事件，在下载过程中有效，下载完成后失效
    public static class NotificationClickReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            Log.d(TAG, " NotificationClickReceiver onReceive");
            if (intent.getAction().equals(DownloadManager.ACTION_NOTIFICATION_CLICKED)
                    && tv_apk_result != null) {
                long[] downIds = intent.getLongArrayExtra(DownloadManager.EXTRA_
NOTIFICATION_CLICK_DOWNLOAD_IDS);
                for (long downId : downIds) {
                    Log.d(TAG, " notify click! id : " + downId + ", mDownloadId=" + mDownloadId);
                    if (downId == mDownloadId) {
                        tv_apk_result.setText(DateUtil.getNowDateTime(null) + " 编号"
                                + downId + "的下载进度条被点击了一下");
                    }
                }
            }
        }
    }
```

上述代码接收并处理了两种下载事件，所以要在 AndroidManifest.xml 中注册对应类的广播信息，具体注册代码如下：

```
<receiver android:name=".DownloadApkActivity$DownloadCompleteReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.DOWNLOAD_COMPLETE" />
    </intent-filter>
</receiver>

<receiver android:name=".DownloadApkActivity$NotificationClickReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.DOWNLOAD_NOTIFICATION_CLICKED" />
    </intent-filter>
</receiver>
```

APK 下载的通知栏效果如图 10-20 和图 10-21 所示。其中，图 10-20 所示为下载进行中的通知栏界面，图 10-21 所示为下载完成后的通知栏界面。

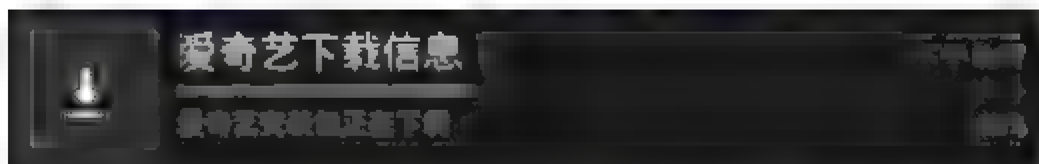


图 10-20　下载进行中的通知栏

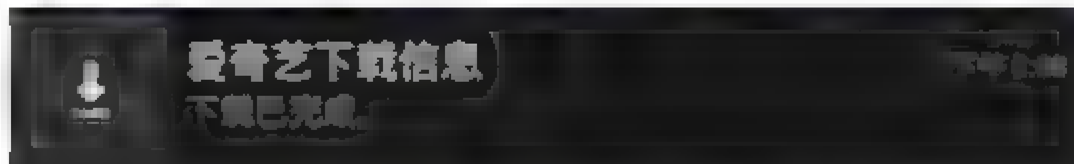


图 10-21　下载完成后的通知栏

不想在通知栏展示下载进度，而是由 App 自身在页面上显示进度也是可行的。下面是在页面上展示下载进度的代码片段：

```
    private Handler mHandler = new Handler();
    private Runnable mRefresh = new Runnable() {
        @Override
        public void run() {
            boolean bFinish = false;
            Query down_query = new Query();
            down_query.setFilterById(mDownloadId);
            Cursor cursor = mDownloadManager.query(down_query);
            if (cursor.moveToFirst()) {
                for (;; cursor.moveToNext()) {
                    int nameIdx = cursor.getColumnIndex(DownloadManager.COLUMN_LOCAL_
FILENAME);
                    int mediaTypeIdx = cursor.getColumnIndex(DownloadManager.COLUMN_
MEDIA_TYPE);
                    int totalSizeIdx = cursor.getColumnIndex(DownloadManager.COLUMN_
TOTAL_SIZE_BYTES);
                    int nowSizeIdx = cursor.getColumnIndex(DownloadManager.COLUMN_
BYTES_DOWNLOADED_SO_FAR);
                    int statusIdx = cursor.getColumnIndex(DownloadManager.COLUMN_STATUS);
                    int progress = (int) (100 * cursor.getLong(nowSizeIdx) / cursor.getLong(totalSizeIdx));
                    if (cursor.getString(nameIdx) == null) {
                        break;
                    }
                    tpc_progress.setProgress(progress, 100);
                    mImagePath = cursor.getString(nameIdx);
                    String desc = "";
                    desc = String.format("%s 文件路径 ： %s\n", desc, cursor.getString(nameIdx));
                    desc = String.format("%s 媒体类型 ： %s\n", desc, cursor.getString(mediaTypeIdx));
                    desc = String.format("%s 文件总大小 ： %d\n", desc, cursor.getLong(totalSizeIdx));
                    desc = String.format("%s 已下载大小 ： %d\n", desc, cursor.getLong(nowSizeIdx));
                    desc = String.format("%s 下载进度 ： %d%%\n", desc, progress);
                    desc = String.format("%s 下载状态 ： %s\n", desc, mStatusMap.get(cursor.
getInt(statusIdx)));
                    tv_image_result.setText(desc);
                    if (progress == 100) {
                        bFinish = true;
                    }
                    if (cursor.isLast() == true) {
                        break;
                    }
```



```
                    }
                }
                cursor.close();
                if (bFinish != true) {
                    mHandler.postDelayed(this, 100);
                } else {
                    sp_image_url.setEnabled(true);
                    tpc_progress.setVisibility(View.INVISIBLE);
                    iv_image_url.setImageURI(Uri.parse(mImagePath));
                }
            }
        };
```

上述代码不在通知栏显示下载进度，即将通知可见类型设置为 VISIBILITY_HIDDEN，此时需要在 AndroidManifest.xml 中加入对应权限，具体的权限配置如下：

```
<!-- 下载时不提示通知栏 -->
<uses-permission android:name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
```

在页面上动态展示图片下载进度的效果如图 10-22 和图 10-23 所示。进度形式采用 10.1 节介绍的文字进度圈，在下载过程中显示带百分比文字的进度圆圈，下载完成后显示已下载的图片。其中，图 10-22 所示为刚开始下载、进度是 4%时的下载页面，此时采用进度圆圈占位；图 10-23 所示为下载完毕的页面，此时占位用的进度圆圈消失，取而代之的是下载到本地的图片。



图 10-22　刚开始下载图片时的进度圈



图 10-23　图片下载完成的界面

10.3.2　文件对话框

下载和上传操作涉及文件的保存和打开，就像电脑上的文件对话框，既可选择文件又可保存文件。然而 Android 没有提供现成的文件对话框控件，我们要自己实现文件对话框。有关对话框的自定义代码可参见第 6 章的“6.3　自定义对话框”，文件对话框的实现走的是另一条路，即利用 DialogFragment 自定义对话框。

还记得第 5 章的“5.3 碎片 Fragment”吧，DialogFragment 其实是碎片 Fragment 的一个子类，生命周期和具体用法可参照 Fragment。当然，Fragment 并非仅有 DialogFragment 一个子类，还有其他几个子类，分别用在某些特殊场合。下面进行简要说明。

- DialogFragment：用于对话框的碎片。对话框的页面构建要写在 onCreateDialog 方法中。
- ListFragment：用于列表的碎片，目的是取代 ListActivity。
- PreferenceFragment：用于参数设置页面的碎片，目的是取代 PreferenceActivity。比如 Android 自带的“系统设置”应用使用了 PreferenceFragment。
- WebViewFragment：用于网页视图的碎片。

由于文件对话框的具体实现代码较长，因此不贴在书上了，有兴趣的读者可自行查看本书下载资源中的相关源码。

文件对话框的展示效果如图 10-24 和图 10-25 所示。其中，图 10-24 所示为保存文件的对话框截图，图 10-25 所示为打开文件的对话框截图。

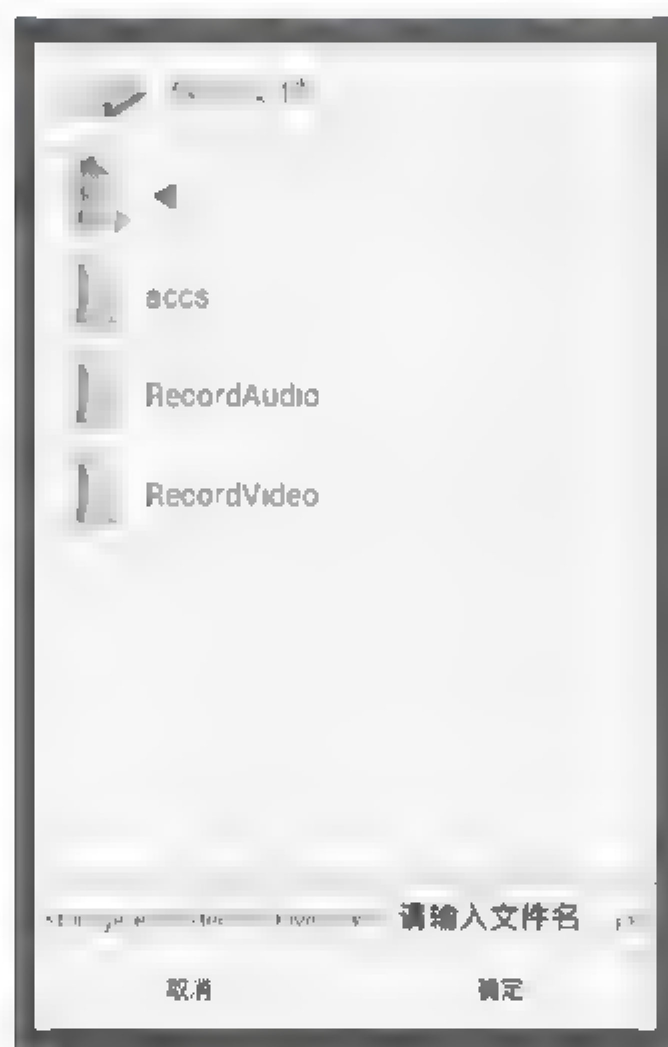


图 10-24 文件保存对话框

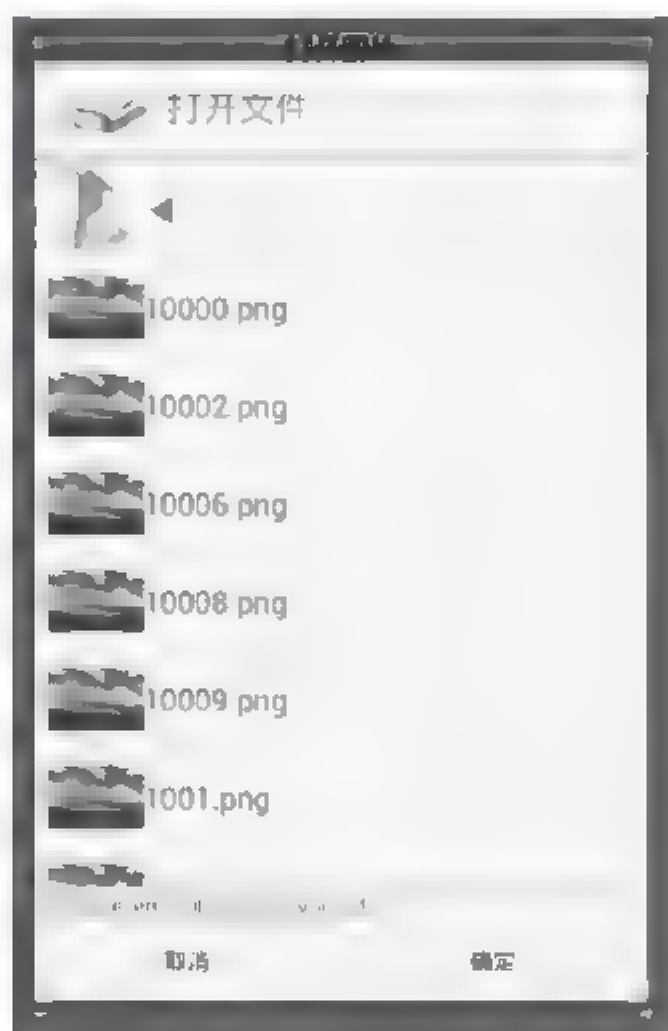


图 10-24 文件打开对话框

考虑到文件对话框是一个通用控件，并且拥有统一风格的图标、文字与尺寸，建议为其单独建一个名为 filedialog 的新模块。其他模块若有用到文件对话框，则可直接导入 filedialog，无须手工复制代码与各类资源。导入 filedialog 的办法是，打开其他模块的编译配置文件 build.gradle，在 dependencies 依赖块中增加如下配置，表示导入 filedialog：

```
compile project(':filedialog')
```

10.3.3 文件上传

与文件下载相比，文件上传的场合不是很多，通常用于上传用户头像、朋友圈发布图片和视频动态等，而且上传文件需要后端服务器配合，容易被开发者忽略。网络通信少不了文件上传，特别是对于社交类 App（如微信、QQ、微博等）来说，上传文件是必不可少的功能，因此有必要掌握文件上传的相关技术。

很可惜，Android 提供了下载管理器 DownloadManager，却没有提供专门的文件上传工具，开发者得自己写代码实现上传功能。简单实现文件上传其实也不难，一样是按照 HTTP 访问的 POST 流程，只是要采取 multipart/form-data 的方式分段传输，并加入分段传输的边界字符串。

下面是通过 HttpURLConnection 上传文件的代码：

```
public class HttpUploadUtil {
    public static String upload(String uploadUrl, String uploadFile) {
        String fileName = "";
        int pos = uploadFile.lastIndexOf("/");
        if (pos >= 0) {
            fileName = uploadFile.substring(pos + 1);
        }
        String end = "\r\n";
        String Hyphens = "--";
        String boundary = "WUm4580jbtwfJhNp7zi1djFEO3wNNm";
        try {
            URL url = new URL(uploadUrl);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setDoInput(true);
            conn.setDoOutput(true);
            conn.setUseCaches(false);
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Connection", "Keep-Alive");
            conn.setRequestProperty("Charset", "UTF-8");
            conn.setRequestProperty("Content-Type", "multipart/form-data;boundary=" + boundary);

            DataOutputStream ds = new DataOutputStream(conn.getOutputStream());
            ds.writeBytes(Hyphens + boundary + end);
            ds.writeBytes("Content-Disposition: form-data; "
                    + "name=\"file1\";filename=\"" + fileName + "\"" + end);
            ds.writeBytes(end);
            FileInputStream fStream = new FileInputStream(uploadFile);
            // 每次写入 1024 字节
            int bufferSize = 1024;
            byte[] buffer = new byte[bufferSize];
            int length = -1;
            // 将文件数据写入缓冲区
            while((length = fStream.read(buffer)) != -1) {
                ds.write(buffer, 0, length);
            }
            ds.writeBytes(end);
            ds.writeBytes(Hyphens + boundary + Hyphens + end);
            fStream.close();
```

```
                ds.flush();
                // 获取返回内容
                InputStream is = conn.getInputStream();
                int ch;
                StringBuffer b = new StringBuffer();
                while ((ch = is.read()) != -1) {
                    b.append((char) ch);
                }
                ds.close();
                return "SUCC";
            } catch (Exception e) {
                e.printStackTrace();
                return "上传失败:" + e.getMessage();
            }
        }
}
```

然后通过异步任务 AsyncTask 调用文件上传功能，文件上传任务的代码如下：

```
public class UploadHttpTask extends AsyncTask<String, Void, String> {
    private final static String TAG = "UploadHttpTask";
    private Context mContext;

    public UploadHttpTask(Context context) {
        super();
        mContext = context;
    }

    @Override
    protected String doInBackground(String... params) {
        String uploadUrl = params[0];
        String filePath = params[1];
        Log.d(TAG, "uploadUrl=" + uploadUrl + ", filePath=" + filePath);
        String result = HttpUploadUtil.upload(uploadUrl, filePath);
        return result;
    }

    @Override
    protected void onPostExecute(String result) {
        mListener.onUploadFinish(result);
    }

    private OnUploadHttpListener mListener;
    public void setOnUploadHttpListener(OnUploadHttpListener listener) {
```

```
            mListener = listener;
        }

        public static interface OnUploadHttpListener {
            public abstract void onUploadFinish(String result);
        }
    }
```

文件上传需要服务器配合，即服务器要开启 HTTP 的上传服务，这涉及服务端开发。正所谓一入 IT 深似海，学了客户端还得会服务端，赶紧找个做 J2EE 的同学帮忙，搭建一下 HTTP 的上传服务器。若一时半刻找不到帮手也没关系，只要读者的笔记本电脑自带无线网卡，就能自己动手、丰衣足食。上传服务器的具体搭建步骤如下：

步骤01 在笔记本电脑上下载并安装“360 免费 WiFi”软件，运行该工具给电脑开启 WIFI 热点，在工具界面上设置 WIFI 名称、用户名、密码等信息。

步骤02 关闭 Windows 系统服务的防火墙。无论是系统自带的 Windows Firewall，还是其他杀毒软件的防火墙，统统关掉；否则手机连不上电脑的 WIFI。

步骤03 打开手机的 WLAN 功能，连接电脑刚开的 WIFI，要求手机能够上网才算连接成功。

步骤04 在笔记本电脑上打开 Eclipse，导入本书附带的文件上传服务端 demo——UploadTest 工程；右击该工程并依次选择 Run As→Run on Server，启动该工程。

步骤05 在命令窗口运行 ipconfig /all，在结果中找到 Microsoft Virtual WiFi Miniport Adapter，红框部分为手机观察到的电脑 IP（如图 10-26 所示），也就是 App 认可的服务器 IP。

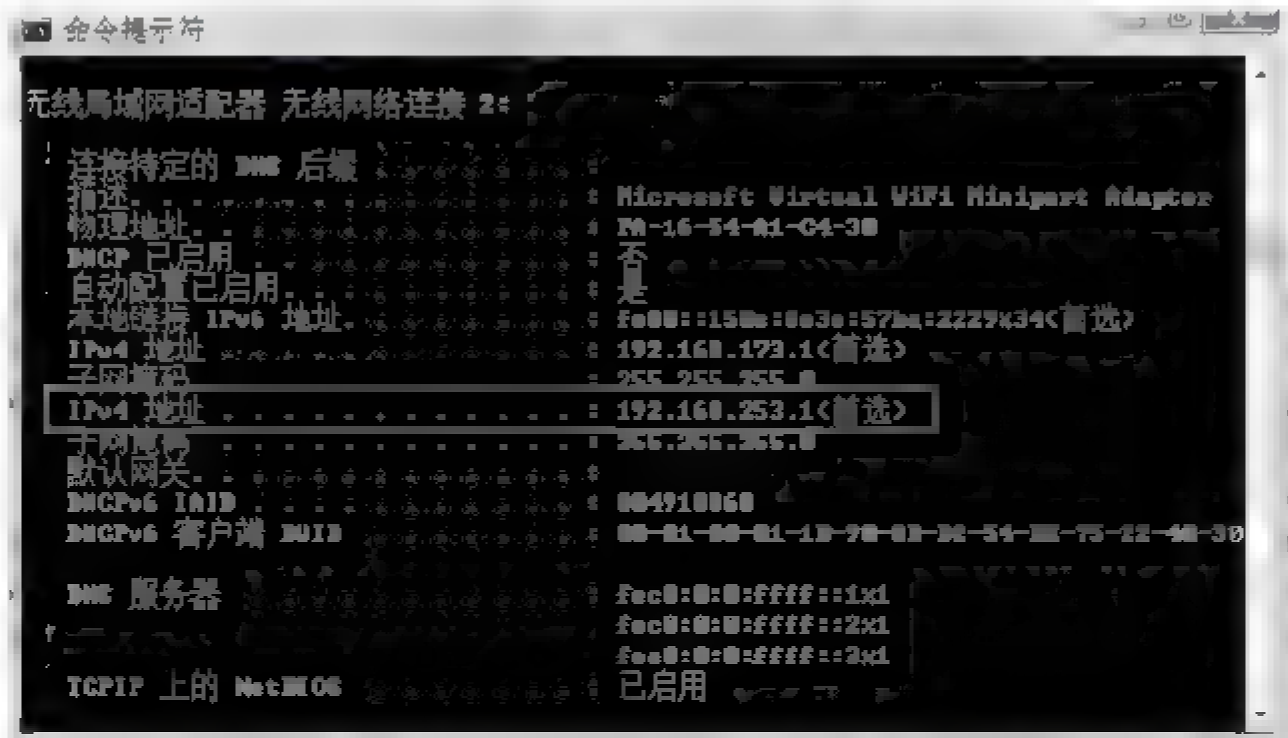


图 10-26 在命令行下面找到的电脑 WIFI 的 IP 地址

在笔记本电脑上搭建好模拟的 HTTP 上传服务器，再修改 App 工程中的上传地址（如 http://192.168.253.1:8080/UploadTest/uploadServlet），然后把 App 安装到手机上，就可以在手机上测试文件上传功能了。

文件上传的效果如图 10-27 所示。倘若上传成功，还应给出服务器对应的文件下载地址，这样才好验证上传成功与否。



图 10-27 文件上传成功的效果图

10.4 套接字 Socket

本节介绍套接字 Socket 的技术手段与具体用途，首先说明如何使用网络地址工具 InetAddress 判断某个网络地址的连通性，然后阐述 Socket 技术在计算机网络中所处的层次、应用方向以及基本用法。

10.4.1 网络地址 InetAddress

有些时候，手机明明可以上网，App 也加了网络访问权限，可是 HTTP 请求某个地址却总是连不上。遇到这种情况，很可能是把对方的地址弄错了，导致尝试连接一个根本连不了的地址。所以有必要在发起请求前检查一下能否与对方地址建立连接。

检查设备自身与某个网络地址的连通性用到了 InetAddress 工具，这是对网络地址的一个封装。下面介绍该工具的主要方法说明。

- getByName：根据主机 IP 或主机名称获取 InetAddress 对象。
- getHostAddress：获取主机的 IP 地址。
- getHostName：获取主机的名称。
- isReachable：判断该地址是否可到达，即是否连通。

下面是检查网络地址能否连通的代码片段：

```
public void onClick(View v) {
    if (v.getId() == R.id.btn_host_name) {
        new CheckThread(et_host_name.getText().toString()).start();
    }
}

private Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        tv_host_name.setText("主机检查结果如下 ：\n"+msg.obj);
    }
};

private class CheckThread extends Thread {
    private String mHostName;
    public CheckThread(String host_name) {
        mHostName = host_name;
    }

    @Override
```

```
            public void run() {
                Message message = Message.obtain();
                try {
                    InetAddress host = InetAddress.getByName(mHostName);
                    boolean isReachable = host.isReachable(5000);
                    String desc = (isReachable)?"可以连接":"无法连接";
                    if (isReachable == true) {
                        desc = String.format("%s\n 主机名为%s\n 主机地址为%s",
                                desc, host.getHostName(), host.getHostAddress());
                    }
                    message.what = 0;
                    message.obj = desc;
                } catch (Exception e) {
                    e.printStackTrace();
                    message.what = -1;
                    message.obj = e.getMessage();
                }
                mHandler.sendMessage(message);
            }
        }
```

检查网络地址连通性的效果如图 10-28 和图 10-29 所示。其中，图 10-28 是根据网站域名检测连通性，图 10-29 是根据 IP 地址检测连通性。



图 10-28　检测域名的连通性结果图



图 10-29　检测 IP 的连通性结果图

10.4.2　Socket 通信

对于程序开发来说，网络通信的基础就是 Socket，不过正因为是基础，所以用起来不容易。计算机网络有一个大名鼎鼎的 TCP/IP 协议，普通用户在电脑上设置本地连接的 IP 时经常看到图 10-30 所示的弹窗，注意红框部分已经很好地描述了 TCP/IP 协议的作用。

TCP/IP 是一个协议组，分为 3 个层次：网络层、传输层和应用层。

- 网络层：包括 IP 协议、ICMP 协议、ARP 协议、RARP 协议和 BOOTP 协议。
- 传输层：包括 TCP 协议和 UDP 协议。
- 应用层：包括 HTTP、FTP、TELNET、SMTP、DNS 等协议。



本章之前提到的网络通信编程其实都是应用层的 HTTP 编程。Socket 属于传输层的技术，API 实现 TCP 协议后即可用于 HTTP 通信，实现 UDP 协议后即可用于 FTP 通信，当然也可以直接在底层进行点对点通信，比如即时通信软件（QQ、微信）就是这样。除了即时通信，Socket 技术也常常用于在线咨询、消息推送等需要实时交互消息的场合。

Android 的 Socket 编程主要使用 Socket 和 ServerSocket 两个类，下面分别进行介绍。

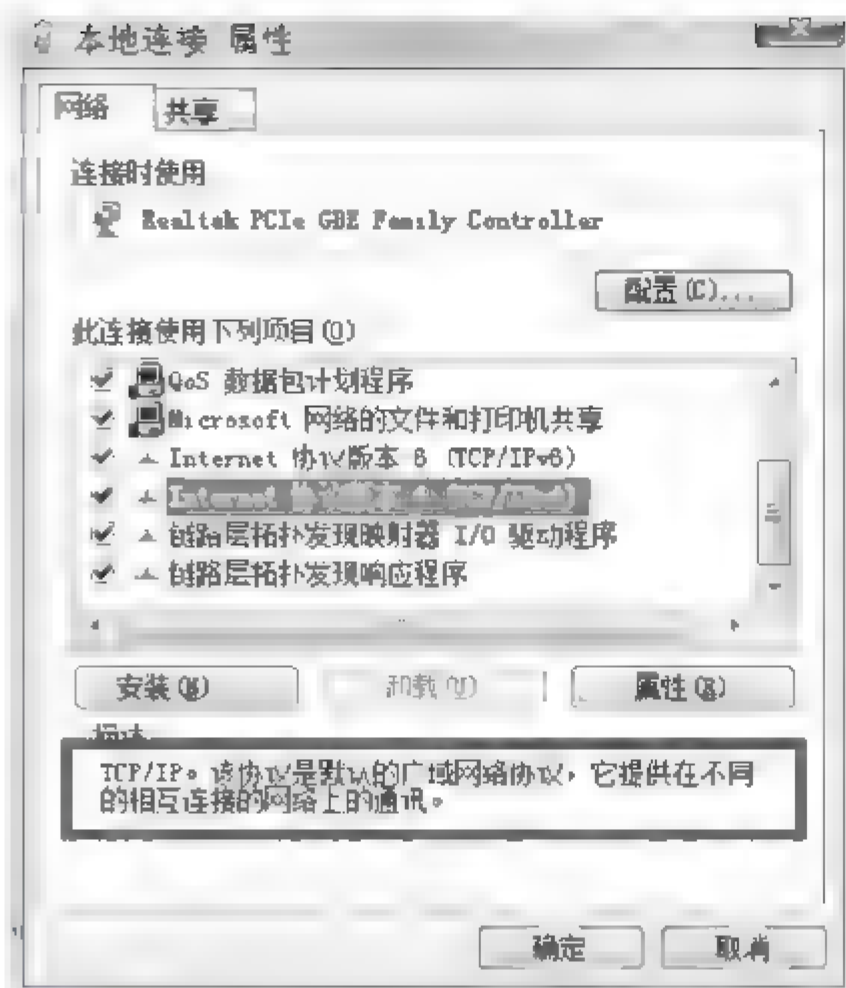


图 10-30 电脑上的本地连接配置页面

1. Socket

Socket 是最常用的工具，客户端和服务端都要用到，描述了两边对套接字（Socket）处理的一般行为。下面介绍 Socket 的主要方法。

- connect：连接指定 IP 和端口。该方法用于客户端连接服务端。
- getInputStream：获取输入流，即自身收到对方发过来的数据。
- getOutputStream：获取输入流，即自身向对方发送的数据。
- getInetAddress：获取网络地址对象。该对象是一个 InetAddress 实例。
- isConnected：判断 socket 是否连上。
- isClosed：判断 socket 是否关闭。
- close：关闭 socket。

2. ServerSocket

ServerSocket 仅用于服务端，在运行时不停地侦听指定端口。下面介绍 ServerSocket 的主要方法。

- 构造函数：指定侦听哪个端口。
- accept：开始接收客户端的连接。有客户端连上时就返回一个 Socket 对象，若要持续侦听连接，则在循环语句中调用该函数。
- getInetAddress：获取网络地址对象。该对象是一个 InetAddress 实例。
- isClosed：判断 socket 服务器是否关闭。
- close：关闭 socket 服务器。

下面通过具体代码演示 Socket 通信的案例，首先在客户端与服务端之间建立 Socket 连接，详细代码如下：

```
public class MessageTransmit implements Runnable {
    private static final String TAG = "MessageTransmit";
    private static final String SOCKET_IP = "192.168.1.5";  // Socket 服务器的 IP，根据实际情况修改
    private static final int SOCKET_PORT = 51000;  // Socket 服务器的端口，根据实际情况修改
    private Socket mSocket;
```

```
    private BufferedReader mReader = null;
    private OutputStream mWriter = null;

    @Override
    public void run() {
        mSocket = new Socket();
        try {
            mSocket.connect(new InetSocketAddress(SOCKET_IP, SOCKET_PORT), 3000);
            mReader = new BufferedReader(new InputStreamReader(mSocket.getInputStream()));
            mWriter = mSocket.getOutputStream();
            new RecvThread().start();  // 启动一条子线程读取服务器的返回数据
            Looper.prepare();
            Looper.loop();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // 定义接收 UI 线程的 Handler 对象，App 向后台服务器发送消息
    public Handler mRecvHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            Log.d(TAG, "handleMessage: "+msg.obj);
            String send_msg = msg.obj.toString()+"\n";
                                        // 换行符相当于回车键，表示“我写好了发出去吧”
            try {
                mWriter.write(send_msg.getBytes("utf8"));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    };

    // 定义消息接收子线程，App 从后台服务器接收消息
    private class RecvThread extends Thread {
        @Override
        public void run() {
            try {
                String content = null;
                while ((content = mReader.readLine()) != null) {    // 读取来自服务器的数据
                    Message msg = Message.obtain();
                    msg.obj = content;
                    SocketActivity.mHandler.sendMessage(msg);
                }
```



```
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
}
```

然后在 Activity 中启动 Socket 连接的线程，等待界面向 Socket 服务器发送消息，并准备接收消息，完整的页面代码如下：

```
public class SocketActivity extends AppCompatActivity implements OnClickListener {
    private EditText et_socket;
    private static TextView tv_socket;
    private MessageTransmit mTransmit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_socket);
        et_socket = (EditText) findViewById(R.id.et_socket);
        tv_socket = (TextView) findViewById(R.id.tv_socket);
        findViewById(R.id.btn_socket).setOnClickListener(this);
        mTransmit = new MessageTransmit();
        new Thread(mTransmit).start();
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_socket) {
            Message msg = Message.obtain();
            msg.obj = et_socket.getText().toString();
            // 注意这里是主线程向分线程发送消息，与 IntentService 的情况类似
            mTransmit.mRecvHandler.sendMessage(msg);
        }
    }

    public static Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            String desc = String.format("%s 收到服务器的应答消息 ：%s",
                    DateUtil.getNowTime(), msg.obj.toString());
            tv_socket.setText(desc);
        }
    };
}
```

最后启动 Socket 服务器（其实一开始就要启动，这样 App 运行时才能马上连上后端服务器），Socket 服务端的源码见本书下载资源 Socket 工程的 TestServer.java，服务器搭建过程参见 10.3 节的“10.3.3 文件上传”末尾部分。

Socket 通信的效果如图 10-31 和图 10-32 所示。其中，图 10-31 所示为准备发送消息时的界面，图 10-32 所示为点击发送按钮后的界面。此时 App 向 Socket 服务器发送消息内容“hello，你好呀”，Socket 服务器立即回复“hi，很高兴认识你”，回复内容已即时显示在 App 页面上。



图 10-31　Socket 消息发送前的界面



图 10-32　Socket 消息发送后的界面

10.5　实战项目：仿手机 QQ 的聊天功能

说到使用最广泛的 App，那无疑是以手机 QQ、微信为代表的即时通信或社交类 App，类似的 App 还有陌陌、旺旺等。这些社交 App 基本都是主打聊天功能，聊天的内容包括文本消息、图片消息、语音消息、视频消息等，其展现内容之丰富、通信手段之多样实为 App 界的翘楚。本章以“仿手机 QQ 的聊天功能”作为实战项目，通过剖析即时通信的相关技术使得读者进一步加深对网络通信开发的理解。

10.5.1　设计思路

手机 QQ 的聊天界面大家再熟悉不过了。不过作为 App 开发者的你，是否能够自己“山寨”一个聊天 App？听起来很高深的样子，自己只是一个初学者啊。有道是世上无难事，只怕有心人，谁说初学者就做不到？下面跟着笔者一步一步往下走，慢慢抽丝剥茧，看看 QQ 内部到底藏着什么“葵花宝典”。

先来两张手机 QQ 的界面截图。图 10-33 所示为联系人频道的好友列表页面；图 10-34 所示为与好友聊天的主页面，文本消息、图片消息、语音消息都在这里发送与接收。

看过了官方 App 的界面，再回来琢磨与本章的网络通信技术有什么关系。也许读者初来乍到，还不太明白，下面给出一个山寨后的效果图，让读者先有一个直观的认识。准备山寨的效果页面如图 10-35 和图 10-36 所示。其中，图 10-35 是山寨后的好友列表页面，图 10-36 是山寨后的聊天页面。

现在看起来，功能相对纯粹了许多，去掉了无关的技术部分，只保留与本章知识点有关的内容。

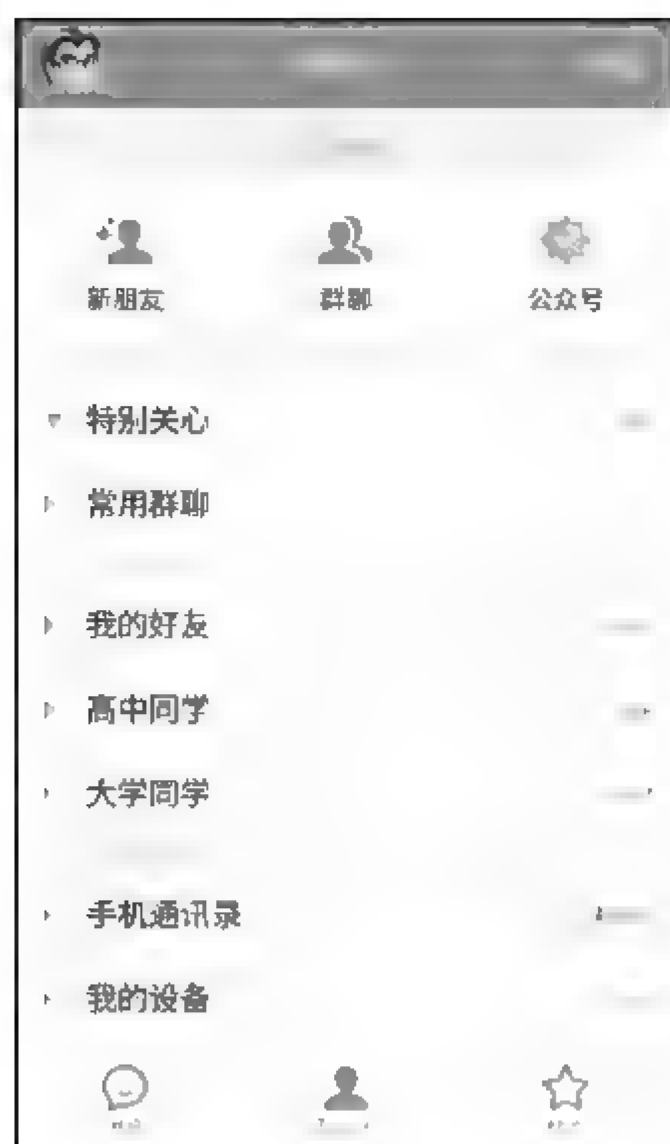
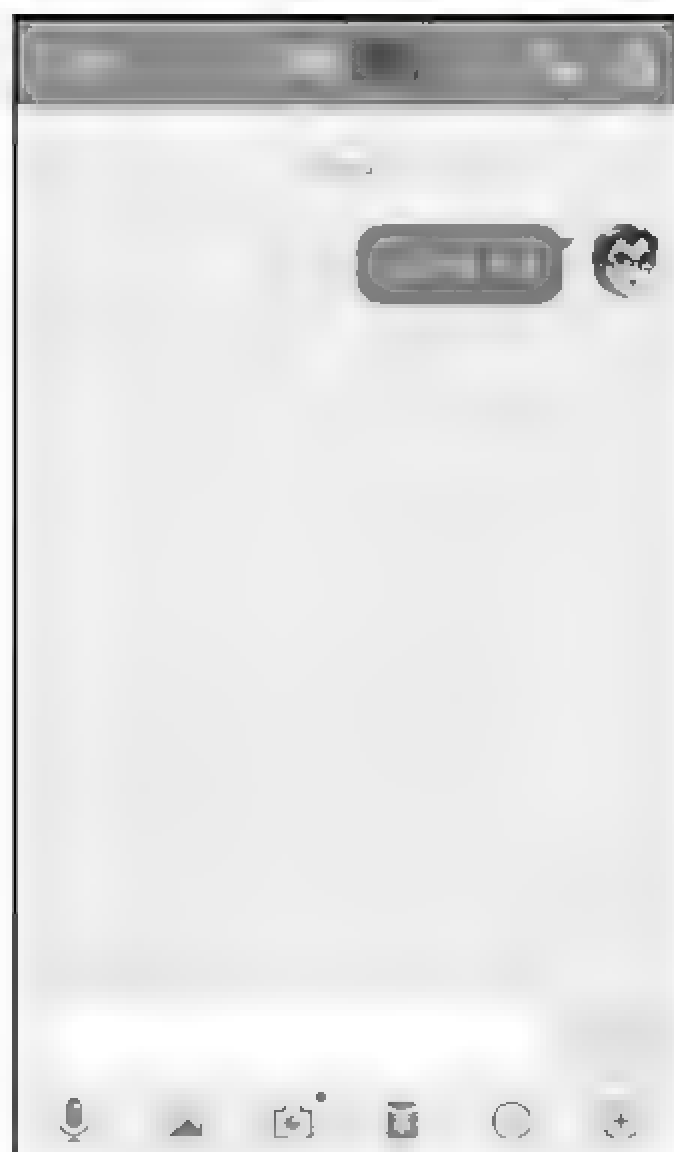


图 10-33　好友列表页面图



10-34　聊天窗口页面



图 10-35　山寨后的好友列表页面

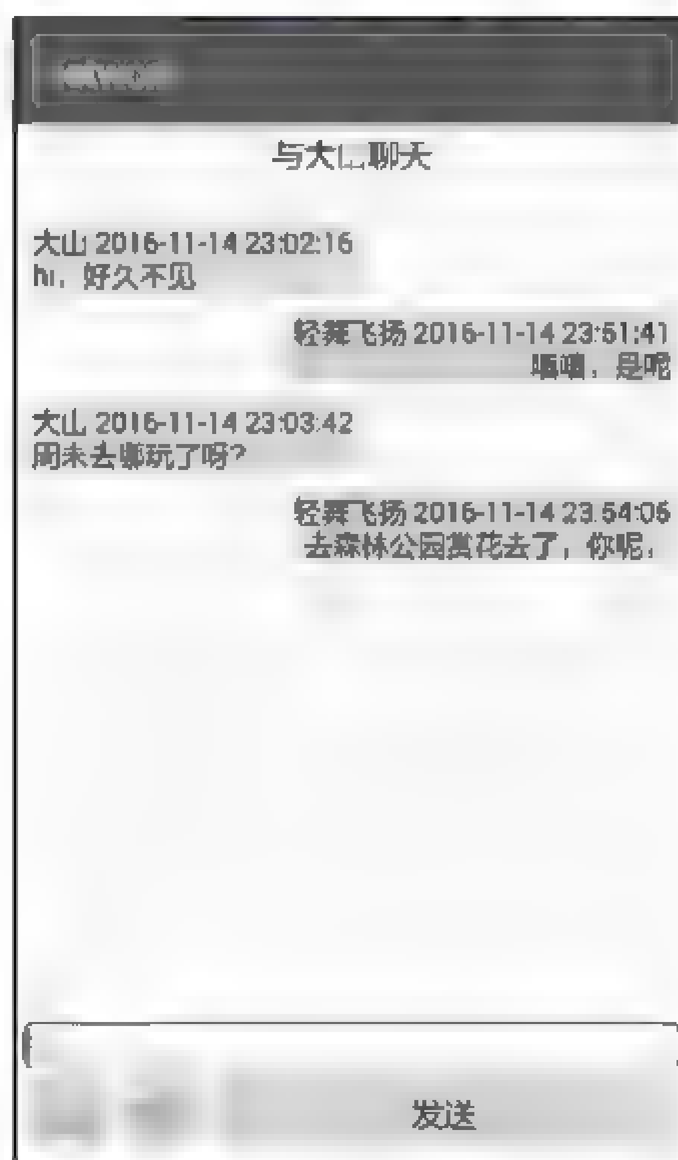


图 10-36　山寨后的聊天窗口页面

下面分析一下效果图涉及的本章的知识点。控件看得见、摸得着，列举并不困难，而网络通信在后台运行，不是马上能够想得到、说得出的，所以不妨尽可能地发挥想像力，无论有没有、能不能实现，先列举出来再说。笔者这里归纳以下 8 个重点。

- 多线程：网络通信必然使用多线程技术。
- HTTP 接口调用：App 向服务器请求全部好友列表，是正规的 HTTP 接口访问。
- HTTP 获取图片：好友的最新头像，因为是小图，所以适合通过 HTTP 协议直接获取图片。

- 文件上传：发送图片消息、发送语音消息时都得把手机上的图片或声音文件上传给服务器。
- 文件下载：对方接收图片和语音消息，很可能从服务器下载图片和声音文件到手机里。
- 文件对话框：选择上传的文件和保存收到的文件都会用到文件对话框。
- 文字进度圆圈：对方接收图片或语音时，聊天窗口往往先显示占位图标，再根据下载进度展示圆弧形式的百分比，让用户知晓消息发送与接收的进展。
- Socket 通信：这个技术是重中之重，因为所有聊天消息都通过 Socket 通信传送给对方。

接下来对 Socket 通信进行补充说明，因为涉及客户端与服务端的交互，所以通信的流程稍微有些复杂。

1. 划分聊天场景的功能点

平常我们看到的 QQ 聊天相关页面有 3 个：登录页面、好友列表页面和聊天页面。因此，对应的 Socket 功能也分为 3 类：

（1）登录与注销。登录操作对应建立 Socket 连接，注销操作对应断开 Socket 连接。

（2）获取在线好友列表。与 Socket 有关的是获取当前在线的好友列表，客户端到服务端查询当前已建立 Socket 连接的好友列表。

（3）发送消息与接收消息。发送与接收消息对应的是 Socket 的数据传输，发送消息操作是客户端 A 向服务端发送 Socket 数据，接收消息操作是服务端将收到的 A 消息向客户端 B 发送 Socket 数据。

2. 在 App 端实现相关功能

（1）至少 3 个聊天相关页面：登录页面、好友列表页面和聊天页面。

（2）一个用于 Socket 通信的线程。由于在 App 运行过程中要保持 Socket 连接，因此 Socket 线程要放在自定义的 Application 类中。

（3）聊天页面向 Socket 线程发送消息的机制，用于登录请求、注销请求、获取在线好友列表请求、发送消息等。

（4）Socket 线程向页面发送消息的机制，用于返回在线好友列表、接收消息等。因为返回消息会分发到不同的页面，所以建议采用广播 Broadcast 传播消息，在好友列表页面和聊天页面各注册一个广播接收器，根据服务器返回的数据刷新在线好友列表和聊天记录。

（5）对于图片消息与声音消息的发送。可先把文件上传到服务器，然后把文件下载地址作为消息文本传给对方；对方 App 收到消息后，根据消息文本中的文件地址把文件下载到本地，再在聊天页面上展示出来。

3. 在服务端启动 Socket 服务器

启动 Socket 服务器后，实现以下功能：

（1）定义一个 Socket 连接的队列，用于保存当前连接的 Socket 请求。

（2）循环侦听指定端口，一旦有新连接进来，就将该连接加入 Socket 队列，并启动新线程为该连接服务。

（3）每个服务线程持续从 Socket 中读取客户端发过来的数据，并对不同请求做相应的处理。

① 如果是登录请求，就标识该 Socket 连接的用户昵称、设备编号、登录时间等信息。

② 如果是注销请求，就断开 Socket 连接，并从 Socket 队列中移除该连接。

③ 如果是获取在线好友列表请求，就遍历 Socket 队列，封装好友列表数据并返回。

④ 如果是发送消息请求，就根据好友的设备编号到Socket队列中查找对应的Socket连接，并向该连接返回消息内容。

4. 定义服务端与客户端之间传输消息的格式

消息包分为包头与包体，包头用于标识操作类型、操作对象、操作时间等基本要素，包体用于存放具体的消息内容（如好友列表、消息文本等）。Socket 通信一般不用 XML 或 JSON 等复杂格式，而是直接用分隔符划分包头、包体以及包头内部的元素。

10.5.2　小知识：可折叠列表视图 ExpandableListView

可折叠列表视图是一种多功能的高级控件，每个子项都可以展开一个孙子列表。点击一个分组（子项），即可展开该分组下的孙子列表；再次点击该分组，即可收起该分组下的孙子列表。如果 LinearLayout 是一维视图，ListView 与 GridView 是二维视图，ExpandableListView 就是三维视图。可折叠列表视图虽然号称高级，但使用的场合不少，常见的业务场景包括好友分组与好友列表、邮件夹分组与邮件列表、订单列表与订单内的商品列表等。

下面是可折叠列表视图的常用方法说明。

- setAdapter：设置适配器。适配器类型为 ExpandableListAdapter。
- expandGroup：展开指定分组。
- collapseGroup：收起指定分组。
- isGroupExpanded：判断指定分组是否展开。
- setSelectedGroup：设置选中的分组。
- setSelectedChild：设置选中的孙子项。
- setGroupIndicator：设置指定分组的指示图像。
- setChildIndicator：设置指定孙子项的指示图像。
- setOnGroupExpandListener：设置分组展开监听器。需实现接口 OnGroupExpandListener 的 onGroupExpand 方法，该方法在点击展开分组时触发。
- setOnGroupCollapseListener：设置分组收起监听器。需实现接口 OnGroupCollapseListener 的 onGroupCollapse 方法，该方法在点击收起分组时触发。
- setOnGroupClickListener：设置分组点击监听器。需实现接口 OnGroupClickListener 的 onGroupClick 方法，该方法在点击分组时触发。
- setOnChildClickListener：设置孙子项点击监听器。需实现接口 OnChildClickListener 的 onChildClick 方法，该方法在点击孙子项时触发。

可折叠列表视图拥有专属的适配器——可折叠列表适配器 ExpandableListAdapter。下面是该适配器经常要重写的 5 个方法。

- getGroupCount：获取分组的个数。
- getChildrenCount：获取孙子项的个数。

- getGroupView：获取指定分组的视图。
- getChildView：获取指定孙子项的视图。
- isChildSelectable：判断孙子项是否允许选择。

下面演示如何使用可折叠列表视图实现电子邮箱的管理功能。首先编写邮箱列表的适配器，详细代码如下（为节省篇幅，省略了没有实际内容的重写方法）：

```
public class MailExpandAdapter implements
ExpandableListAdapter,OnGroupClickListener,OnChildClickListener {
    private LayoutInflater mInflater;
    private Context mContext;
    private ArrayList<MailBox> mBoxList;

    public MailExpandAdapter(Context context, ArrayList<MailBox> box_list) {
        mInflater = LayoutInflater.from(context);
        mContext = context;
        mBoxList = box_list;
    }

    @Override
    public int getGroupCount() {
        return mBoxList.size();
    }

    @Override
    public int getChildrenCount(int groupPosition) {
        return mBoxList.get(groupPosition).mail_list.size();
    }

    @Override
    public View getGroupView(int groupPosition, boolean isExpanded, View convertView, ViewGroup
parent) {
        ViewHolderBox holder = null;
        if (convertView == null) {
            holder = new ViewHolderBox();
            convertView = mInflater.inflate(R.layout.item_box, null);
            holder.iv_box = (ImageView) convertView.findViewById(R.id.iv_box);
            holder.tv_box = (TextView) convertView.findViewById(R.id.tv_box);
            holder.tv_count = (TextView) convertView.findViewById(R.id.tv_count);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolderBox) convertView.getTag();
        }
        MailBox box = mBoxList.get(groupPosition);
```



```
        holder.iv_box.setImageResource(box.box_icon);
        holder.tv_box.setText(box.box_title);
        holder.tv_count.setText(box.mail_list.size()+"封邮件");
        return convertView;
    }

    @Override
    public View getChildView(final int groupPosition, final int childPosition,
                boolean isLastChild, View convertView, ViewGroup parent) {
        ViewHolderMail holder = null;
        if (convertView == null) {
            holder = new ViewHolderMail();
            convertView = mInflater.inflate(R.layout.item_mail, null);
            holder.ck_mail = (CheckBox) convertView.findViewById(R.id.ck_mail);
            holder.tv_date = (TextView) convertView.findViewById(R.id.tv_date);
            convertView.setTag(holder);
        } else {
            holder = (ViewHolderMail) convertView.getTag();
        }
        MailItem item = mBoxList.get(groupPosition).mail_list.get(childPosition);
        holder.ck_mail.setText(item.mail_title);
        holder.ck_mail.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                MailBox box = mBoxList.get(groupPosition);
                MailItem item = box.mail_list.get(childPosition);
                String desc = String.format("您点击了%s，标题是%s", box.box_title, item.mail_title);
                Toast.makeText(mContext, desc, Toast.LENGTH_SHORT).show();
            }
        });
        holder.tv_date.setText(item.mail_date);
        return convertView;
    }

    @Override
    public boolean isChildSelectable(int groupPosition, int childPosition) {
        return true;   //如果子条目需要响应点击事件，这里就要返回 true
    }

    public final class ViewHolderBox {
        public ImageView iv_box;
        public TextView tv_box;
        public TextView tv_count;
```

```
    }

    public final class ViewHolderMail {
        public CheckBox ck_mail;
        public TextView tv_date;
    }

    @Override
    public boolean onChildClick(ExpandableListView parent, View v, int groupPosition, int childPosition,
long id) {
        ViewHolderMail holder = (ViewHolderMail) v.getTag();
        holder.ck_mail.setChecked(!(holder.ck_mail.isChecked()));
        return true;
    }

    @Override
    public boolean onGroupClick(ExpandableListView parent, View v, int groupPosition, long id) {
        String desc = String.format("您点击了%s", mBoxList.get(groupPosition).box_title);
        Toast.makeText(mContext, desc, Toast.LENGTH_SHORT).show();
        return false;   //如果返回 true，就不会展示子列表
    }
}
```

然后在页面代码中给可折叠列表视图设置对应的适配器对象，具体的代码片段如下：

```
    private void initMailBox() {
        ExpandableListView elv_list = (ExpandableListView) findViewById(R.id.elv_list);
        final ArrayList<MailBox> box_list = new ArrayList<MailBox>();
        box_list.add(new MailBox(R.drawable.mail_folder_inbox, "收件箱", getRecvMail()));
        box_list.add(new MailBox(R.drawable.mail_folder_outbox, "发件箱", getSentMail()));
        box_list.add(new MailBox(R.drawable.mail_folder_draft, "草稿箱", getDraftMail()));
        box_list.add(new MailBox(R.drawable.mail_folder_recycle, "废件箱", getRecycleMail()));
        MailExpandAdapter adapter = new MailExpandAdapter(this, box_list);
        elv_list.setAdapter(adapter);
        elv_list.setOnChildClickListener(adapter);
        elv_list.setOnGroupClickListener(adapter);
        elv_list.expandGroup(0);   //默认展开第一个邮件夹
    }
```

电子邮箱的展示效果如图 10-37 和图 10-38 所示。其中，图 10-37 所示为展开收件箱时的初始界面，图 10-38 所示为点击收起收件箱后，点击展开发件箱时的界面。





图 10-37 展开收件箱时的初始界面



图 10-38 点击展开发件箱时的界面

可折叠列表视图有时会出现孙子项不响应点击事件的问题，可能是某个环节没有正确设置。要让孙子项正常响应点击事件，需满足下面 3 个条件：

（1）可折叠列表适配器的 isChildSelectable 方法要返回 true。

（2）可折叠列表视图的对象要调用 setOnChildClickListener 方法注册孙子项的点击监听器，并重写该监听器的 onChildClick 方法。

（3）孙子项目中若存在 Button、EditText 等默认占用焦点的控件，则要去除焦点占用，即调用这些控件的 setFocusable 和 setFocusableInTouchMode 方法，并设置为 false。也可参照第 5 章“5.2.2 列表视图 ListView”中处理子项抢占焦点的做法，给列表项布局文件的根节点加上 descendantFocusability 属性，声明在列表项范围内剥夺下级控件的抢占权利，代码如下：

```
android:descendantFocusability="blocksDescendants"
```

10.5.3 代码示例

编码方面需要注意以下两点：

（1）访问网络、文件上传与下载时，要在 AndroidManifest.xml 添加对应的权限配置：

```
<!-- 互联网 -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- 查看网络状态 -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- SD 卡 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

（2）AndroidManifest.xml 的 application 节点注意补充 android:name=".MainApplication"。另外，要注册在线好友列表获取和消息接收的广播接收器，注册代码如下：

```
<receiver android:name=".ChatMainActivity$RecvMsgReceiver" >
    <intent-filter>
        <action android:name="com.example.network.RECV_MSG" />
    </intent-filter>
</receiver>

<receiver android:name=".QQContactActivity$GetListReceiver" >
    <intent-filter>
        <action android:name="com.example.network.GET_LIST" />
    </intent-filter>
</receiver>
```

因为网络通信需要服务端配合，所以服务器方面需要实现 3 个后台功能。

（1）文件上传功能，源码参见本书下载资源 UploadTest 工程里的 UploadServlet.java。
（2）获取好友列表接口，源码参见本书下载资源 UploadTest 工程里的 QueryFriend.java。
（3）Socket服务器，源码参见本书下载资源的Socket工程，主程序入口在ChatServer.java中。

实战项目不但要在真机上测试，而且要开启服务端程序，这样才能真刀真枪地模拟即时通信的真实场景。首先在服务器上分别启动 UploadTest 工程和 Socket 工程，然后准备两台手机分别安装实战项目编译后的 App（注意：App 代码中的 URL 服务地址必须是正确的服务器 IP，并且手机与服务器在同一个网段），接着两台手机都运行实战项目的聊天 App，在图 10-39 所示的登录页面输入昵称，点击登录按钮，进入好友列表页面。

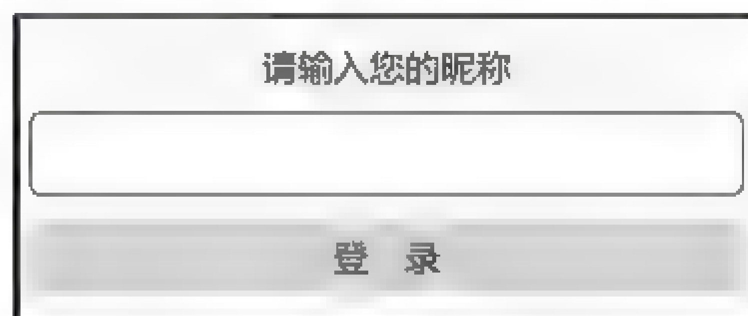


图 10-39 实战项目的登录页面

好友列表的页面效果如图 10-40 和图 10-41 所示。其中，图 10-40 所示为展开在线好友分组时的好友列表界面，可以看到手机 A 的登录昵称是“轻舞飞扬”，手机 B 的登录昵称是“大山”；图 10-41 所示为展开亲戚分组时的好友列表界面。



图 10-40 展开在线好友分组时的界面



图 10-41 展开亲戚分组时的列表界面

两台手机都点击对方昵称，进入聊天主页面，页面下方有文本编辑框，可发送文本消息。左下角有图片的图标按钮，点击即可选择图片并发送图片消息；图片按钮右边是声音的图标按钮，点击即可选择音频文件并发送声音消息。为了看起来更逼真，消息窗口采用对方消息靠左对齐、我方消息靠右对齐的布局，并给双方消息着不同的背景色。对于文本消息来说，双方消息窗口直接展示文字内容就可以了；对于图片消息来说，消息窗口应展示图片的缩略图，用户点击缩略图后，再展示图片的大图；对于声音消息来说，消息窗口只展示声音图标，一旦用户点击声音图标，系统就开始播放对应的声音文件。

具体测试的聊天效果如图 10-42～图 10-45 所示。其中，图 10-42 和图 10-43 所示为手机 A（昵称“轻舞飞扬”）的聊天窗口，图 10-44 和图 10-45 所示为手机 B（昵称“大山”）的聊天窗口。

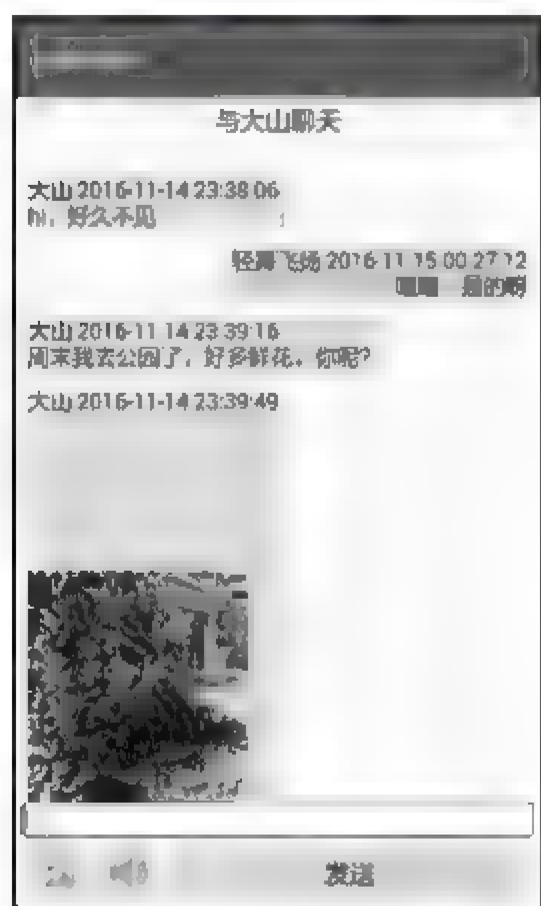


图 10-42 与大山的聊天窗口截图 1



图 10-43 与大山的聊天窗口截图 2



图 10-44 与轻舞飞扬的聊天窗口截图 1

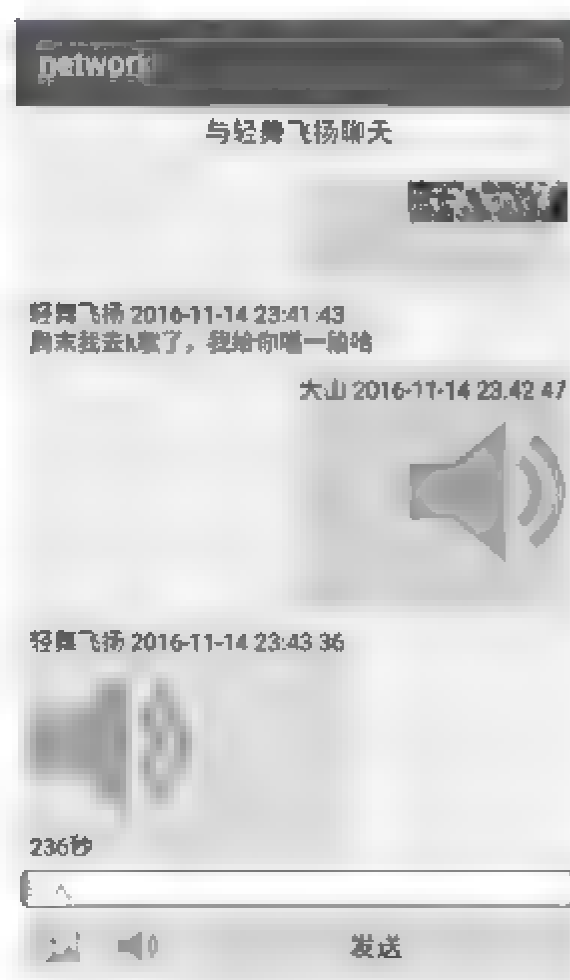


图 10-45 与轻舞飞扬的聊天窗口截图 2

至此，实战项目仿照手机 QQ 基本实现了聊天功能的常用操作，包括实时刷新在线好友列表、发送与接收文本消息、发送与接收图片消息、发送与接收声音消息等。当然，本项目尚有

若干待完善的地方，有兴趣的读者可自行补充修改，完善点主要有以下 3 点：

（1）发送图片与声音时，目前采用文件对话框选择具体文件。其实可参照第 9 章的设备操作，现场拍照或现场录音后，把照片和录音文件直接传给对方。另外，可增加对视频消息的发送与接收处理。

（2）目前聊天消息没有保存到本地数据库，因此下次打开对方的聊天窗口无法查看之前的聊天记录。可参照第 4 章的 SQLite 数据库操作把聊天消息保存到 SQLite 中，这样每次打开聊天窗口时都会到数据库中查找并显示历史聊天记录。

（3）把第 9 章的实战项目“仿微信的发现功能”集成到本章的实战项目中，增强这个聊天 App 的实用性和趣味性。

还等什么呢？快快行动起来，打造一个专属的即时通信 App 吧！

下面是好友列表页面的代码，更多聊天代码参见本书下载资源的源码：

```
public class QQContactActivity extends AppCompatActivity implements
        OnClickListener, OnQueryFriendListener {
    private final static String TAG = "QQContactActivity";
    private static Context mContext;
    private static ExpandableListView elv_friend;
    private static ArrayList<FriendGroup> mGroupList;
    private static FriendGroup mGroupOnline = new FriendGroup();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_qq_contact);
        Toolbar tl_head = (Toolbar) findViewById(R.id.tl_head);
        tl_head.setTitle(getResources().getString(R.string.menu_second));
        setSupportActionBar(tl_head);
        tl_head.setNavigationOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                finish();
            }
        });
        mContext = getApplicationContext();
        mGroupOnline.title = "在线好友";
        mGroupList = new ArrayList<FriendGroup>();
        elv_friend = (ExpandableListView) findViewById(R.id.elv_friend);
        findViewById(R.id.btn_refresh).setOnClickListener(this);
        QueryFriendTask queryTask = new QueryFriendTask(this);
        queryTask.setOnQueryFriendListener(this);
        queryTask.execute();
```

```
}

@Override
public void onQueryFriend(String resp) {
    try {
        JSONObject obj = new JSONObject(resp);
        JSONArray groupArray = obj.getJSONArray("group_list");
        for (int i=0; i<groupArray.length(); i++) {
            JSONObject groupObj = groupArray.getJSONObject(i);
            FriendGroup group = new FriendGroup();
            group.title = groupObj.getString("title");
            JSONArray friendArray = groupObj.getJSONArray("friend_list");
            for (int j=0; j<friendArray.length(); j++) {
                JSONObject friendObj = friendArray.getJSONObject(j);
                Friend friend = new Friend("", friendObj.getString("name"), "");
                group.friend_list.add(friend);
            }
            mGroupList.add(group);
        }
        showAllFriend();
    } catch (JSONException e) {
        e.printStackTrace();
        Toast.makeText(this, "获取全部好友列表出错 ："+e.getMessage(),
                Toast.LENGTH_SHORT).show();
    }
}

@Override
protected void onResume() {
    mHandler.postDelayed(mRefresh, 500);
    super.onResume();
}

@Override
protected void onDestroy() {
    MainApplication.getInstance().sendAction(ClientThread.LOGOUT, "", "");
    super.onDestroy();
}

private Handler mHandler = new Handler();
private Runnable mRefresh = new Runnable() {
    @Override
    public void run() {
```

```
                MainApplication.getInstance().sendAction(ClientThread.GETLIST, "", "");
            }
        };

        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.btn_refresh) {
                mHandler.post(mRefresh);
            }
        }

        public static class GetListReceiver extends BroadcastReceiver {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent != null) {
                    Log.d(TAG, "onReceive");
                    String content = intent.getStringExtra(ClientThread.CONTENT);
                    if (mContext != null && content != null && content.length() > 0) {
                        showFriendOnline(content);
                    }
                }
            }
        }

        private static void showFriendOnline(String content) {
            int pos = content.indexOf(ClientThread.SPLIT_LINE);
            String head = content.substring(0, pos);
            String body = content.substring(pos + 1);
            String[] splitArray = head.split(ClientThread.SPLIT_ITEM);
            if (splitArray[0].equals(ClientThread.GETLIST)) {
                String[] bodyArray = body.split("\\|");
                ArrayList<Friend> friendList = new ArrayList<Friend>();
                for (int i = 0; i < bodyArray.length; i++) {
                    String[] itemArray = bodyArray[i].split(ClientThread.SPLIT_ITEM);
                    if (bodyArray[i].length() > 0 && itemArray != null && itemArray.length >= 3) {
                        friendList.add(new Friend(itemArray[0], itemArray[1], itemArray[2]));
                    }
                }
                mGroupOnline.friend_list = friendList;
                showAllFriend();
            } else {
                String hint = String.format("%s\n%s", splitArray[0], body);
                Toast.makeText(mContext, hint, Toast.LENGTH_SHORT).show();
```

```
            }
        }

        private static void showAllFriend() {
            ArrayList<FriendGroup> all_group = new ArrayList<FriendGroup>();
            all_group.add(mGroupOnline);
            all_group.addAll(mGroupList);
            FriendExpandAdapter adapter = new FriendExpandAdapter(mContext, all_group);
            elv_friend.setAdapter(adapter);
            elv_friend.setOnChildClickListener(adapter);
            elv_friend.setOnGroupClickListener(adapter);
            elv_friend.expandGroup(0);   //默认展开在线好友
        }
    }
```

10.6 小　结

本章主要介绍了 App 开发用到的网络通信相关技术，包括多线程的工作机制和用法（消息传递、进度对话框、异步任务、异步服务）、HTTP 接口访问的方式（网络连接检查、移动数据格式、HTTP 接口调用、HTTP 图片获取）、文件上传和下载的实现与用法（下载管理器、文件对话框、文件上传）、套接字的应用（网络地址、Socket 通信）。最后设计了一个实战项目“仿手机 QQ 的聊天功能”，详细阐述了即时通信技术的原理与设计思路。在该项目的 App 编码中，采用了本章介绍的大部分网络通信知识，实现了文本消息、图片消息、声音消息的发送与接收。另外，介绍了如何使用可折叠列表视图。

通过本章的学习，读者应该能够掌握以下 4 种开发技能：

（1）学会在合适的场合使用多线程技术。
（2）学会 HTTP 方式的接口调用与图片获取。
（3）学会管理文件上传和下载操作。
（4）学会运用 Socket 通信技术进行聊天应用的开发。



事　件

本章介绍 App 开发常见的一些事件处理技术，主要包括如何检测并接管按键事件，如何对触摸事件进行分发、拦截与处理，如何实现手势检测与飞掠视图的联合运用，如何正确避免手势冲突的意外状况。最后结合本章所学的知识演示一个实战项目“抠图神器——美图变变”的设计与实现。

11.1　按键事件

本节介绍 App 开发对按键事件的检测与处理，首先说明如何检测控件对象的按键事件；然后说明如何检测活动页面的物理按键，并以返回键为例阐述“再按一次返回键退出”的功能实现；最后以音量调节对话框为例，介绍如何接管音量按键的处理。

11.1.1　检测软键盘

一般不对手机上的输入按键进行处理，直接由系统按照默认情况操作。当然有时为了改善用户体验，需要让 App 拦截按键事件，并进行额外处理。在第 3 章介绍编辑框 EditText 的用法时提到监控输入字符中的回车键，一旦发现用户敲了回车键，就将焦点自动移到下一个控件，而不是在编辑框输入回车换行。当时的字符输入拦截采用注册文本观测器 TextWatcher 实现，但该监听器只适用于编辑框控件，无法用于其他控件。因此，若想让其他控件能够监听按键操作，则要另外调用控件对象的 setOnKeyListener 方法设置按键监听器，并实现监听器接口 OnKeyListener 的 onKey 方法。

要监控按键事件，首先得知道每个按键的编码，这样才能根据不同的编码值进行相应的处理。按键编码的取值说明见表 11-1。这里注意，监听器 OnKeyListener 只会检测控制键，不会检测文本键（字母、数字、标点等）。

表11-1　按键编码的取值说明

按键编码	KeyEvent 类的按键名称	说明
3	KEYCODE_HOME	主页键（未开放给普通 App）
4	KEYCODE_BACK	返回键（后退键）
24	KEYCODE_VOLUME_UP	加大音量键
25	KEYCODE_VOLUME_DOWN	减小音量键
26	KEYCODE_POWER	电源键（未开放给普通 App）
66	KEYCODE_ENTER	回车键
67	KEYCODE_DEL	删除键（退格键）
82	KEYCODE_MENU	菜单键
84	KEYCODE_SEARCH	搜索键
187	KEYCODE_APP_SWITCH	任务键（未开放给普通 App）

实际监控结果显示，每次按控制键时，onKey 方法都会收到两次重复编码的按键事件，这是因为该方法把每次按键都分成按下与松开两个动作，所以一次按键变成了两个按键动作。解决这个问题的办法很简单，就是只监控按下动作（KeyEvent.ACTION_DOWN）的按键事件，不监控松开动作（KeyEvent.ACTION_UP）的按键事件。

下面是使用软键盘监听器的代码：

```
public class KeySoftActivity extends AppCompatActivity implements OnKeyListener {
    private EditText et_soft;
    private TextView tv_result;
    private String desc = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_key_soft);
        et_soft = (EditText) findViewById(R.id.et_soft);
        tv_result = (TextView) findViewById(R.id.tv_result);
        et_soft.setOnKeyListener(this);
    }

    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (event.getAction()==KeyEvent.ACTION_DOWN) {
            desc = String.format("%s 输入的软按键编码是%d,动作是按下", desc, keyCode);
            if (keyCode == KeyEvent.KEYCODE_ENTER) {
                desc = String.format("%s, 按键为回车键", desc);
            } else if (keyCode == KeyEvent.KEYCODE_DEL) {
                desc = String.format("%s, 按键为删除键", desc);
            } else if (keyCode == KeyEvent.KEYCODE_BACK) {
                desc = String.format("%s, 按键为返回键", desc);
                mHandler.postDelayed(mFinish, 3000);
            } else if (keyCode == KeyEvent.KEYCODE_MENU) {
                desc = String.format("%s, 按键为菜单键", desc);
            } else if (keyCode == KeyEvent.KEYCODE_VOLUME_UP) {
                desc = String.format("%s, 按键为加大音量键", desc);
            } else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {
                desc = String.format("%s, 按键为减小音量键", desc);
            }
            desc = desc + "\n";
            tv_result.setText(desc);
            return true;
        } else {
            return false;    //返回 true 表示处理完了不再输入该字符，返回 false 表示输入该字符
        }
    }

    private Handler mHandler = new Handler();
    private Runnable mFinish = new Runnable() {
        @Override
```

```
            public void run() {
                finish();
            }
        };
    }
```

上述代码的按键效果如图 11-1 所示。虽然按键编码表存在首页键、任务键、电源键的定义，但这 3 个键并不开放给普通 App，普通 App 也不应该拦截这些按键事件。

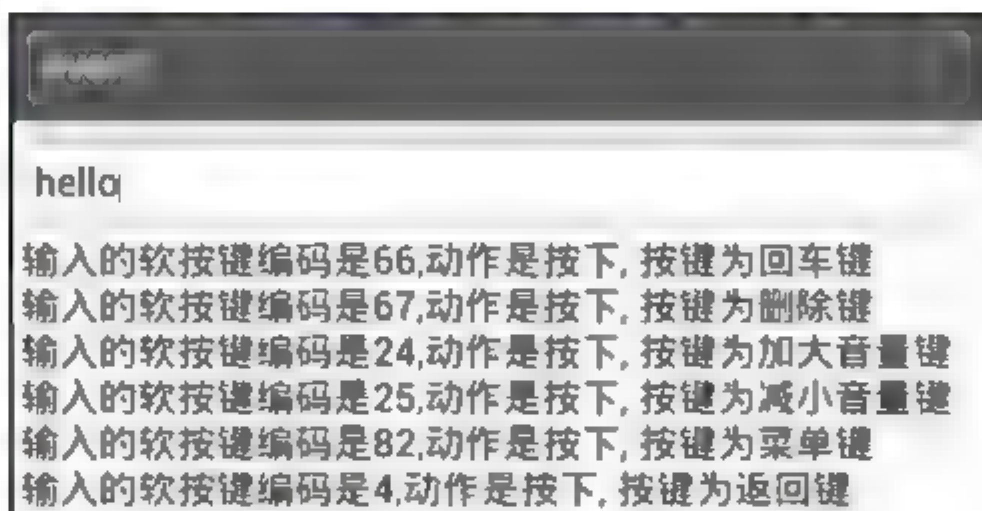


图 11-1　软键盘的检测结果

11.1.2　检测物理按键

除了给控件注册按键监听器外，还可以直接在活动页面上检测物理按键，即重写 Activity 的 onKeyDown 方法。onKeyDown 方法的使用与前面的 onKey 方法类似，拥有按键编码与按键事件 KeyEvent 两个参数，当然这两个方法也存在不同之处，具体说明如下：

（1）onKeyDown 只能在 Activity 代码中使用，而 onKey 只要有可注册的控件就能使用。

（2）onKeyDown 只能检测物理按键，无法检测输入法按键（如回车键、删除键等），而 onKey 可同时检测两类按键。

（3）onKeyDown 不区分按下与松开两个动作，而 onKey 区分这两个动作。

下面是启用物理按键监听的代码片段：

```
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        desc = String.format("%s 物理按键的编码是%d", desc, keyCode);
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            desc = String.format("%s，按键为返回键", desc);
            mHandler.postDelayed(mFinish, 3000);
        } else if (keyCode == KeyEvent.KEYCODE_MENU) {
            desc = String.format("%s，按键为菜单键", desc);
        } else if (keyCode == KeyEvent.KEYCODE_VOLUME_UP) {
            desc = String.format("%s，按键为加大音量键", desc);
        } else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {
            desc = String.format("%s，按键为减小音量键", desc);
        }
        desc = desc + "\n";
```

```
        tv_result.setText(desc);
        return true;   //返回 true 表示不再响应系统动作，返回 false 表示继续响应系统动作
    }

    private Handler mHandler = new Handler();
    private Runnable mFinish = new Runnable() {
        @Override
        public void run() {
            finish();
        }
    };
```

物理按键的监听效果如图 11-2 所示。对于目前的智能手机来说，普通 App 只可检测 4 个物理按键事件，即菜单键、返回键、加大音量键和减小音量键。

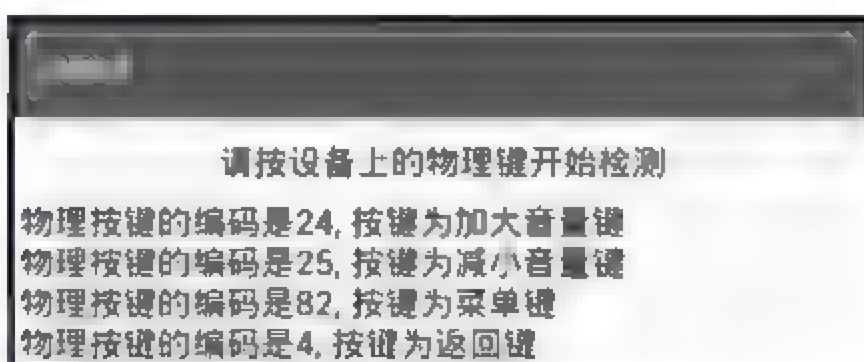


图 11-2　物理按键的检测结果

检测物理按键最常见的应用是淘宝主页的“再按一次返回键退出”，在 App 首页按返回键，系统默认的做法是直接退出该 App。然而用户有可能不小心按了返回键，并非想退出该 App，因此这里加一个小提示，等待用户再次按返回键才会确认退出意图，并执行退出操作。

“再按一次返回键退出”的实现代码很简单，在 onKeyDown 方法中拦截返回键即可，具体代码如下：

```
    private boolean bExit = false;
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_BACK) {
            if (bExit) {
                finish();
            }
            bExit = true;
            Toast.makeText(this, "再按一次返回键退出!", Toast.LENGTH_SHORT).show();
            return true;
        } else {
            return super.onKeyDown(keyCode, event);
        }
    }
```

重写 Activity 代码的 onBackPressed 方法可实现同样的效果，该方法专门响应按返回键事件，具体代码如下：

```
    private boolean bExit = false;
    @Override
    public void onBackPressed() {
```

```
        if (bExit) {
            finish();
            return;
        }
        bExit = true;
        Toast.makeText(this, "再按一次返回键退出!", Toast.LENGTH_SHORT).show();
    }
```

该功能的界面效果如图 11-3 所示。这是一个提示小窗口，在淘宝主页按返回键时能够看到。



图 11-3　“再按一次返回键退出”的提示窗口

11.1.3　音量调节对话框

除了检测回车键与返回键，音量键也常常需要拦截。第 9 章提到 Android 有 6 类铃音，分别是通话音、系统音、铃音、媒体音、闹钟音和通知音，不过音量键只有加大与减少两个键，当用户按音量增加键时，App 怎么知道用户希望加大哪类铃音的音量呢？

要解决这个问题，最好是弹出一个对话框，让用户选择希望调节的铃音类型，并显示拖动条，方便用户把音量一次调整到位，不必连续按增加键或减小键。自定义音量对话框还有一个好处，即允许定制对话框的界面风格与显示位置，这在播放音乐和播放电影时尤其适用。

因为自定义对话框的代码不在 Activity 中，所以无法通过 onKeyDown 方法检测按键，只能给拖动条注册按键监听器 OnKeyListener。自定义音量调节对话框的代码如下：

```
public class VolumeDialog implements OnSeekBarChangeListener, OnKeyListener {
    private Dialog dialog;
    private View view;
    private SeekBar sb_music;
    private AudioManager mAudioMgr;
    private int MUSIC = AudioManager.STREAM_MUSIC;
    private int mMaxVolume;
    private int mNowVolume;

    public VolumeDialog(Context context) {
        mAudioMgr = (AudioManager) context.getSystemService(Context.AUDIO_SERVICE);
        mMaxVolume = mAudioMgr.getStreamMaxVolume(MUSIC);
        mNowVolume = mAudioMgr.getStreamVolume(MUSIC);
        view = LayoutInflater.from(context).inflate(R.layout.dialog_volume, null);
        dialog = new Dialog(context, R.style.VolumeDialog);
        sb_music = (SeekBar) view.findViewById(R.id.sb_music);
        sb_music.setOnSeekBarChangeListener(this);
        sb_music.setProgress(sb_music.getMax() * mNowVolume/mMaxVolume);
```

```
    }

    public void show() {
        dialog.getWindow().setContentView(view);
        dialog.getWindow().setLayout(
                LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT);
        dialog.show();
        sb_music.setFocusable(true);
        sb_music.setFocusableInTouchMode(true);
        sb_music.setOnKeyListener(this);
    }

    public void dismiss() {
        if (dialog != null && dialog.isShowing()) {
            dialog.dismiss();
        }
    }

    public boolean isShowing() {
        if (dialog != null) {
            return dialog.isShowing();
        } else {
            return false;
        }
    }

    public void adjustVolume(int direction, boolean fromActivity) {
        if (direction == AudioManager.ADJUST_RAISE) {
            mNowVolume += 1;
        } else {
            mNowVolume -= 1;
        }
        sb_music.setProgress(sb_music.getMax() * mNowVolume/mMaxVolume);
        mAudioMgr.adjustStreamVolume(MUSIC, direction, AudioManager.FLAG_PLAY_SOUND);
        if (mListener != null && fromActivity!=true) {
            mListener.onVolumeAdjust(mNowVolume);
        }
        close();
    }

    private void close() {
        mHandler.removeCallbacks(mClose);
        mHandler.postDelayed(mClose, 2000);
```

```
    }

    private Handler mHandler = new Handler();
    private Runnable mClose = new Runnable() {
        @Override
        public void run() {
            dismiss();
        }
    };

    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) {
    }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) {
        mNowVolume = mMaxVolume * seekBar.getProgress()/seekBar.getMax();
        mAudioMgr.setStreamVolume(MUSIC, mNowVolume, AudioManager.FLAG_PLAY_SOUND);
        if (mListener != null) {
            mListener.onVolumeAdjust(mNowVolume);
        }
        close();
    }

    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_VOLUME_UP && event.getAction()==KeyEvent.
ACTION_DOWN) {
            adjustVolume(AudioManager.ADJUST_RAISE, false);
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN && event.getAction()==
KeyEvent.ACTION_DOWN) {
            adjustVolume(AudioManager.ADJUST_LOWER, false);
            return true;
        } else {
            return false;
        }
    }
```

```
        private VolumeAdjustListener mListener;
        public void setVolumeAdjustListener(VolumeAdjustListener listener) {
            mListener = listener;
        }

        public static interface VolumeAdjustListener {
            public abstract void onVolumeAdjust(int volume);
        }
    }
```

在页面代码中通过检测音量增加键和减小键弹出音量对话框，代码如下：

```
    public class VolumeSetActivity extends AppCompatActivity implements VolumeAdjustListener {
        private TextView tv_volume;
        private VolumeDialog dialog;
        private AudioManager mAudioMgr;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_volume_set);
            tv_volume = (TextView) findViewById(R.id.tv_volume);
            mAudioMgr = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        }

        @Override
        public boolean onKeyDown(int keyCode, KeyEvent event) {
            if (keyCode == KeyEvent.KEYCODE_VOLUME_UP && event.getAction()==KeyEvent.
ACTION_DOWN) {
                showVolumeDialog(AudioManager.ADJUST_RAISE);
                return true;
            } else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN && event.getAction()==
KeyEvent.ACTION_DOWN) {
                showVolumeDialog(AudioManager.ADJUST_LOWER);
                return true;
            } else if (keyCode == KeyEvent.KEYCODE_BACK) {
                finish();
                return false;
            } else {
                return false;
            }
        }

        private void showVolumeDialog(int direction) {
```

```
            if (dialog==null || dialog.isShowing()!=true) {
                dialog = new VolumeDialog(this);
                dialog.setVolumeAdjustListener(this);
                dialog.show();
            }
            dialog.adjustVolume(direction, true);
            onVolumeAdjust(mAudioMgr.getStreamVolume(AudioManager.STREAM_MUSIC));
        }

        @Override
        public void onVolumeAdjust(int volume) {
            tv_volume.setText("调节后的音乐音量大小为 ："+volume);
        }
    }
```

音量调节对话框的效果如图 11-4 和图 11-5 所示。其中，图 11-4 所示为在主页面按音量增加键时弹出音量对话框的界面；用户把对话框上的拖动条向左拉，以大幅减小音乐音量，此时的音量对话框界面如图 11-5 所示。

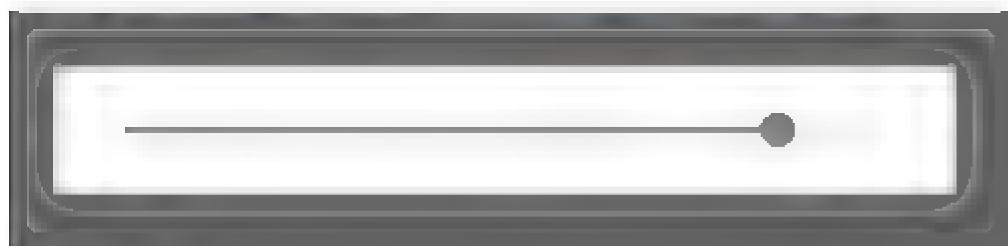


图 11-4　按音量增加键弹出对话框

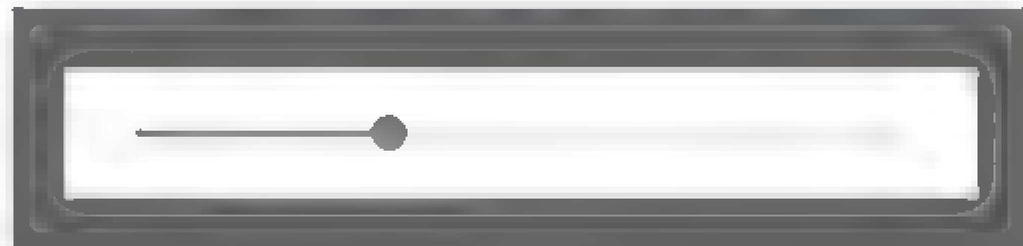


图 11-5　把对话框上的拖动条往左拉

11.2　触摸事件

本节介绍对屏幕触摸事件的相关处理，首先说明手势事件的分发流程，包括 3 个手势方法、3 类手势执行者、派发与拦截处理；然后说明手势事件的具体用法，包括单点触摸和多点触控；最后阐述一个手势触摸的具体应用——手写签名功能的实现。

11.2.1　手势事件的分发流程

智能手机的一大革命性技术是把屏幕变为可触摸设备，既可用于信息输出（显示界面）又可用于信息输入（检测用户的触摸行为）。为方便开发者使用，Android 已经自动识别特定的几种触摸手势，包括按钮的点击事件（OnClickListener）、长按事件（OnLongClickListener）、滚动视图 ScrollView 的上下滚动事件、翻页视图 ViewPager 的左右翻页事件等。不过对于 App 的高级开发来说，系统自带的几个固定手势显然无法满足丰富多变的业务需求。这时就要求开发者深入了解触摸行为的流程与方法，并在合适的场合接管触摸行为，进行符合需求的事件处理。

与手势事件有关的方法主要有 3 个（按执行顺序排列）：dispatchTouchEvent、onInterceptTouchEvent 和 onTouchEvent。

- dispatchTouchEvent：进行事件分发处理，返回结果表示该事件是否需要分发。默认返回 true 表示分发给下级视图，由下级视图处理该手势，不过最终是否分发成功还得根据 onInterceptTouchEvent 方法的拦截判断结果；返回 false 表示不分发，此时必须实现自身的 onTouchEvent 方法，否则该手势将不会得到处理。
- onInterceptTouchEvent：进行事件拦截处理，返回结果表示当前容器是否需要拦截该事件。返回 true 表示予以拦截，该手势不会分发给下级视图，此时必须实现自身的 onTouchEvent 方法，否则该手势将不会得到处理；默认返回 false 表示不拦截，该手势会分发给下级视图进行后续处理。
- onTouchEvent：进行事件触摸处理，返回结果表示该事件是否处理完毕。返回 true 表示处理完毕，无须处理上级视图的 onTouchEvent 方法，一路返回结束流程；返回 false 表示该手势事件尚未完成，返回继续处理上级视图的 onTouchEvent 方法，然后根据上级 onTouchEvent 方法的返回值判断直接结束或由上上级处理。

上述手势方法的执行者有 3 个（按执行顺序排列）：页面类、容器类和控件类。

- 页面类：包括 Activity 及其派生类。页面类可操作 dispatchTouchEvent 和 onTouchEvent 两种方法。
- 容器类：包括从 ViewGroup 类派生出的各类容器，如各种布局 Layout，以及 ListView、GridView、Spinner、ViewPager、RecyclerView、Toolbar 等。容器类可操作 dispatchTouchEvent、onInterceptTouchEvent 和 onTouchEvent 三种方法。
- 控件类：包括从 View 类派生的各类控件，如 TextView、ImageView、Button 等。控件类可操作 dispatchTouchEvent 和 onTouchEvent 两种方法。

可以看出，只有容器类才能操作 onInterceptTouchEvent 方法，这是因为该方法用于拦截发往下层视图的事件，而控件类已经位于底层，只能被拦截，不能拦截别人。页面类不拥有下层视图，所以不能操作 onInterceptTouchEvent 方法。

以上涉及 3 个手势方法和 3 种手势执行者，其中手势流程的排列组合千变万化，并不容易解释清楚。对于实际开发来说，真正需要处理的组合并不多，所以只要把常见的几种组合搞清楚，就能应付大部分开发工作。

首先是页面类的手势处理，其 dispatchTouchEvent 必须返回 true，因为如果不分发，页面上的视图就无法处理手势。至于页面类的 onTouchEvent，基本没什么作用，因为手势动作由具体视图处理，页面直接处理手势没什么意义。所以页面类的手势处理可以不用关心，直接略过。

其次是控件类的手势处理，其 dispatchTouchEvent 没有任何作用，因为控件下面没有下级视图，无所谓分不分发。至于控件类的 onTouchEvent，如果要进行手势处理，就需要自定义一个控件，重写自定义类中的 onTouchEvent 方法；如果不想自定义控件，就直接调用控件对象的 setOnTouchListener 方法，注册一个触摸监听器 OnTouchListener，并实现该监听器的 onTouch 方法。所以控件类的手势处理只需关心 onTouchEvent 方法。

最后是容器类的手势处理，这才是真正要深入了解的地方。容器类的 dispatchTouchEvent 与 onInterceptTouchEvent 两个方法都能决定是否将手势交给下级视图处理。为了避免手势响应冲突，一般要重写 dispatchTouchEvent 方法或 onInterceptTouchEvent 方法。这两个方法之间的

区别可以这么理解：前者是大领导，只管派发任务，不会自己做事情；后者是小领导，尽管有拦截的权利，不过也得自己做点事情，比如处理纠纷。容器类的 onTouchEvent 近乎摆设，因为需要拦截的在前面已经拦截了，需要处理的在下级视图已经处理了，很少会兜一大圈在这儿处理。

经过上面的详细分析，常见的手势处理方法有下面 3 种。

- 容器类的 dispatchTouchEvent 方法：控制事件的分发，决定把手势交给谁处理。
- 容器类的 onInterceptTouchEvent 方法：控制事件的拦截，决定是否要把手势交给下级视图处理。
- 控件类的 onTouchEvent 方法：进行手势事件的具体处理。

下面是一个不派发事件的自定义布局代码：

```
public class NotDispatchLayout extends LinearLayout {
    public NotDispatchLayout(Context context) {
        super(context);
    }

    public NotDispatchLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean dispatchTouchEvent(MotionEvent ev) {
        if (mListener != null) {
            mListener.onNotDispatch();
        }
        return false;   // 一般容器默认返回 true，即允许分发给下级
    }

    private NotDispatchListener mListener;
    public void setNotDispatchListener(NotDispatchListener listener) {
        mListener = listener;
    }

    public static interface NotDispatchListener {
        public abstract void onNotDispatch();
    }
}
```

活动页面实现的 onNotDispatch 方法代码如下：

```
    public void onNotDispatch() {
        desc_no = String.format("%s%s 触摸动作未分发，按钮点击不了了\n"
                    , desc_no, DateUtil.getNowTime());
```

```
            tv_dispatch_no.setText(desc_no);
        }
```

不派发事件的处理效果如图 11-6 和图 11-7 所示。其中，图 11-6 的上面部分为正常布局，此时按钮可正常响应点击事件；图 11-7 的下面部分为不派发布局，此时按钮不会响应点击事件，取而代之的是执行不派发布局的 onNotDispatch 方法。

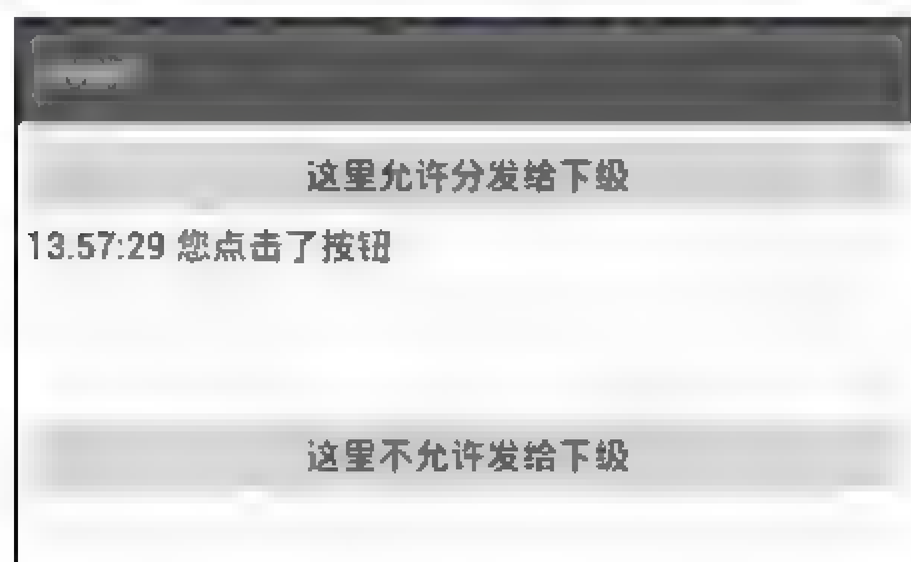


图 11-6　正常布局允许分发事件

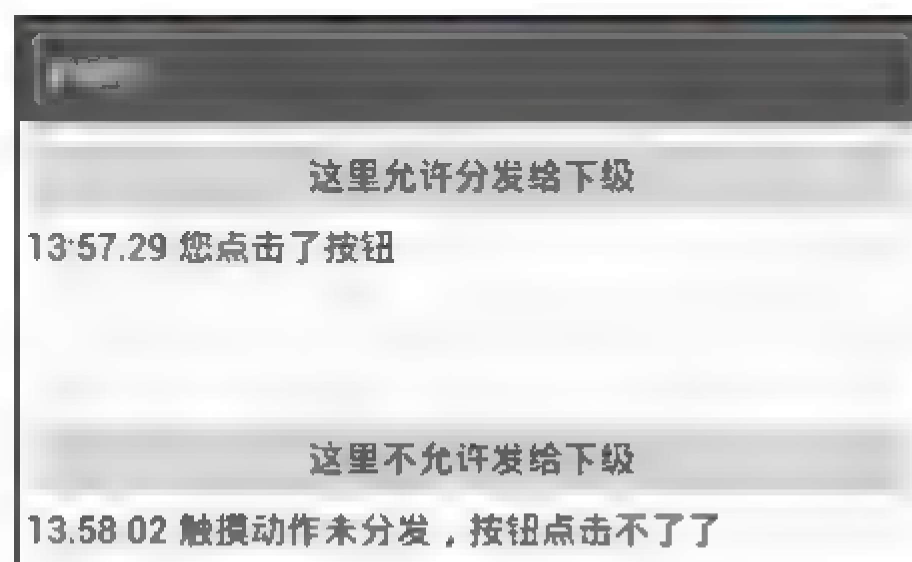


图 11-7　不派发布局未分发事件

再来看看拦截事件的自定义布局代码：

```
public class InterceptLayout extends LinearLayout {
    public InterceptLayout(Context context) {
        super(context);
    }

    public InterceptLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {
        if (mListener != null) {
            mListener.onIntercept();
        }
        return true;    // 一般容器默认返回 false（不拦截），不过滚动视图 ScrollView 会拦截
    }

    private InterceptListener mListener;
    public void setInterceptListener(InterceptListener listener) {
        mListener = listener;
    }

    public static interface InterceptListener {
        public abstract void onIntercept();
    }
}
```

活动页面实现的 onIntercept 方法代码如下：



```
public void onIntercept() {
    desc_yes = String.format("%s%s 触摸动作被拦截，按钮点击不了了\n", desc_yes,
            DateUtil.getNowTime());
    tv_intercept_yes.setText(desc_yes);
}
```

拦截事件的处理效果如图 11-8 和图 11-9 所示。其中，图 11-8 的上面部分为正常布局，此时按钮可正常响应点击事件；图 11-9 的下面部分为拦截布局，此时按钮不会响应点击事件，取而代之的是执行拦截布局的 onIntercept 方法。

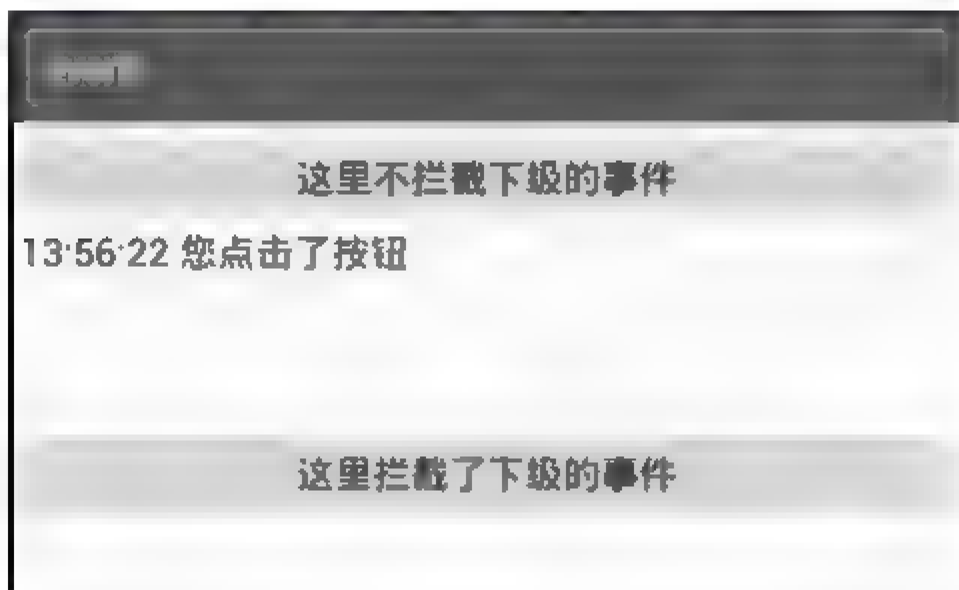


图 11-8　正常布局不拦截事件

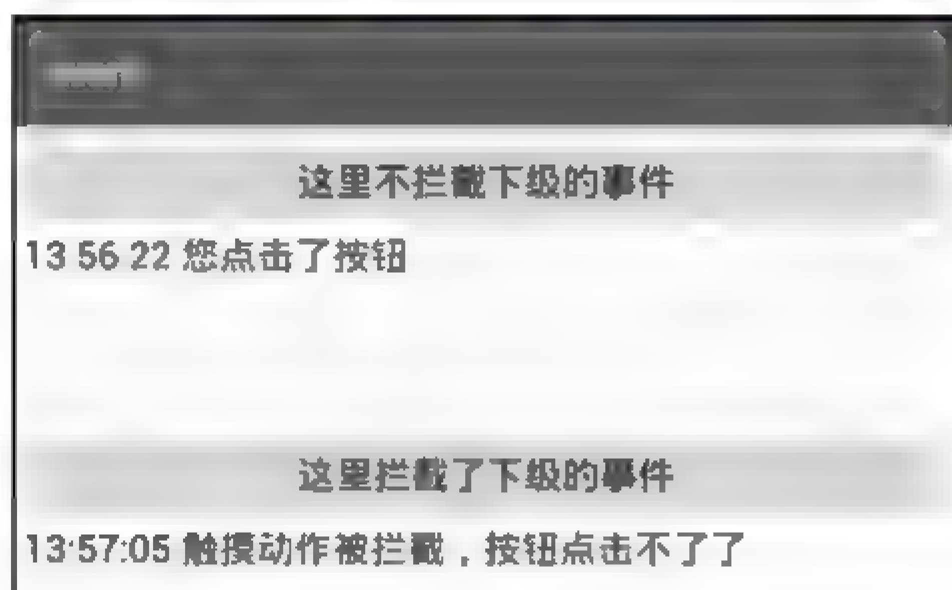


图 11-9　拦截布局拦截事件

11.2.2　手势事件处理 MotionEvent

dispatchTouchEvent、onInterceptTouchEvent 和 onTouchEvent 的输入参数都是手势事件 MotionEvent，其中包含触摸动作的所有信息，各种手势操作都得到 MotionEvent 中获取信息并进行判断处理。

下面是 MotionEvent 的常用方法说明。

- getAction：获取当前的动作类型。动作类型的取值说明见表 11-2。

表11-2　动作类型的取值说明

MotionEvent 类的动作类型	说明
ACTION_DOWN	按下动作
ACTION_UP	提起动作
ACTION_MOVE	移动动作
ACTION_CANCEL	取消动作
ACTION_OUTSIDE	移出边界动作
ACTION_POINTER_DOWN	第二个点的按下动作，用于多点触控的判断
ACTION_POINTER_UP	第二个点的提起动作，用于多点触控的判断
ACTION_MASK	动作掩码，与原动作类型进行"与"（&）操作后获得多点触控信息

- getEventTime：获取事件时间（从开机到现在的毫秒数）。
- getX：获取在控件内部的相对横坐标。
- getY：获取在控件内部的相对纵坐标。

- getRawX：获取在屏幕上的绝对横坐标。
- getRawY：获取在屏幕上的绝对纵坐标。
- getPressure：获取触摸的压力大小。
- getPointerCount：获取触控点的数量，如果为 2 就表示有两个手指同时按压屏幕。如果触控点数目大于 1，坐标相关方法就可输入整型编号，表示获取第几个触控点的坐标信息。

下面是演示单点触摸的页面代码：

```
public class TouchSingleActivity extends AppCompatActivity {
    private TextView tv_touch;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_touch_single);
        tv_touch = (TextView) findViewById(R.id.tv_touch);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int seconds = (int) (event.getEventTime() / 1000);   // 从开机到现在的毫秒数
        int hour = seconds / 3600;
        int minute = seconds % 3600 / 60;
        int second = seconds % 60;
        Date date = new Date(event.getEventTime());
        String desc = String.format("动作发生时间 ：开机距离现在%02d:%02d:%02d",
                hour, minute, second);
        desc = String.format("%s\n 动作名称是 ：", desc);
        int action = event.getAction();
        if (action == event.ACTION_DOWN) {
            desc = String.format("%s 按下", desc);
        } else if (action == event.ACTION_MOVE) {
            desc = String.format("%s 移动", desc);
        } else if (action == event.ACTION_UP) {
            desc = String.format("%s 提起", desc);
        } else if (action == event.ACTION_CANCEL) {
            desc = String.format("%s 取消", desc);
        }
        desc = String.format("%s\n 动作发生位置是 ：横坐标%f，纵坐标%f",
                desc, event.getX(), event.getY());
        tv_touch.setText(desc);
        return super.onTouchEvent(event);
    }
}
```

单点触摸的效果如图 11-10、图 11-11、图 11-12 所示。其中，图 11-10 所示为手势按下时的检测界面，图 11-11 所示为手势移动时的检测界面，图 11-12 所示为手势提起时的检测界面。

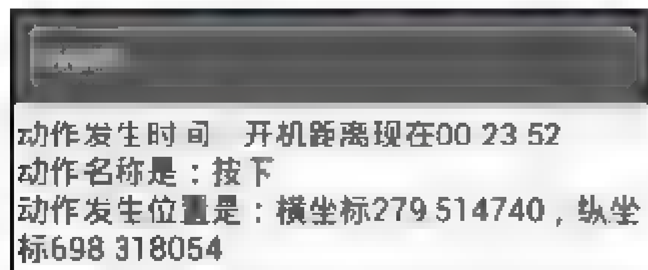


图 11-10　手势按下时的界面



图 11-11　手势移动时的界面

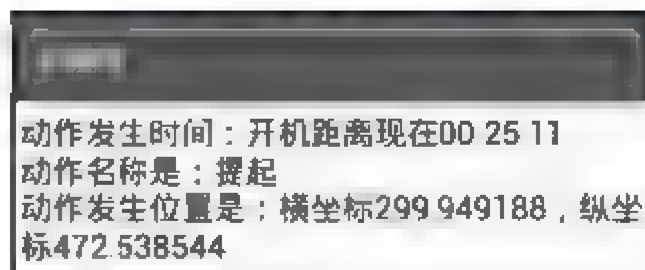


图 11-12　手势提起时的界面

除了单点触摸，智能手机还普遍支持多点触控，即响应两个及以上手指同时按压屏幕。多点触控可用于操纵图像的缩放与旋转操作，以及需要多点处理的游戏界面。

下面是演示多点触控的页面代码：

```
public class TouchMultipleActivity extends AppCompatActivity {
    private TextView tv_touch_main;
    private TextView tv_touch_secondary;
    private boolean bSecondaryPressed = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_touch_multiple);
        tv_touch_main = (TextView) findViewById(R.id.tv_touch_main);
        tv_touch_secondary = (TextView) findViewById(R.id.tv_touch_secondary);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int seconds = (int) (event.getEventTime() / 1000);
        int hour = seconds / 3600;
        int minute = seconds % 3600 / 60;
        int second = seconds % 60;
        Date date = new Date(event.getEventTime());
        String desc_main = String.format("主要动作发生时间 ：开机距离现在%02d:%02d:%02d\n%s",
                hour, minute, second, "主要动作名称是 ：");
        String desc_secondary = "";
        int action = event.getAction() & MotionEvent.ACTION_MASK;
        if (action == event.ACTION_DOWN) {
            desc_main = String.format("%s 按下", desc_main);
        } else if (action == event.ACTION_MOVE) {
            desc_main = String.format("%s 移动", desc_main);
            if (bSecondaryPressed == true) {
                desc_secondary = String.format("%s 次要动作名称是 ：移动", desc_secondary);
            }
```

```
            } else if (action == event.ACTION_UP) {
                desc_main = String.format("%s 提起", desc_main);
            } else if (action == event.ACTION_CANCEL) {
                desc_main = String.format("%s 取消", desc_main);
            } else if (action == event.ACTION_POINTER_DOWN) {
                bSecondaryPressed = true;
                desc_secondary = String.format("%s 次要动作名称是 ： 按下", desc_secondary);
            } else if (action == event.ACTION_POINTER_UP) {
                bSecondaryPressed = false;
                desc_secondary = String.format("%s 次要动作名称是 ： 提起", desc_secondary);
            }
            desc_main = String.format("%s\n 主要动作发生位置是 ： 横坐标%f，纵坐标%f",
                    desc_main, event.getX(), event.getY());
            tv_touch_main.setText(desc_main);
            if (bSecondaryPressed==true || desc_secondary.length()>0) {
                desc_secondary = String.format("%s\n 次要动作发生位置是 ： 横坐标%f，纵坐标%f",
                        desc_secondary, event.getX(1), event.getY(1));
                tv_touch_secondary.setText(desc_secondary);
            }
            return super.onTouchEvent(event);
        }
    }
```

多点触控的效果如图 11-13 和图 11-14 所示。其中，图 11-13 所示为两个手指一起按下时的检测界面，图 11-14 所示为两个手指一齐提起时的检测界面。



图 11-13　两个手指一齐按下时的界面



图 11-14　两个手指一齐提起时的界面

11.2.3　手写签名

为了加深对触摸事件的认识，接下来我们通过实现一个手写签名控件进一步理解手势处理的应用场合。

手写签名的原理是把手机屏幕当作画板，把用户手指当作画笔，手指在屏幕上划来划去，屏幕就会显示手指的移动轨迹，就像画笔在画板上写字一样。实现手写签名需要结合绘图的路径工具 Path，在有按下动作时调用 Path 对象的 moveTo 方法，将路径起始点移到触摸点；在有移动操作时调用 Path 对象的 quadTo 方法，将记录本次触摸点与上次触摸点之间的路径；在有移动操作与提起动作时调用 Canvas 对象的 drawPath 方法，将本次触摸轨迹绘制在画布上。

手写签名控件的自定义代码主要片段如下：

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.drawBitmap(cachebBitmap, 0, 0, null);
    canvas.drawPath(path, paint);          //这个是需要的，最近一次路径保存在这里
}

private float mLastX, mLastY;
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        path.moveTo(event.getX(), event.getY());
        pos.firstX = event.getX();
        pos.firstY = event.getY();
        break;
    case MotionEvent.ACTION_MOVE:
        path.quadTo(mLastX, mLastY, event.getX(), event.getY());
        pos.nextX = event.getX();
        pos.nextY = event.getY();
        pathArray.add(pos);
        pos = new PathPosition();
        pos.firstX = event.getX();
        pos.firstY = event.getY();
        break;
    case MotionEvent.ACTION_UP:
        cacheCanvas.drawPath(path, paint);
        path.reset();
        break;
    }
    mLastX = event.getX();
    mLastY = event.getY();
    invalidate();
    return true;
}
```

手写签名的效果如图 11-15 和图 11-16 所示。其中，图 11-15 所示为写到一半的签名界面，图 11-16 所示为签名完成的界面。

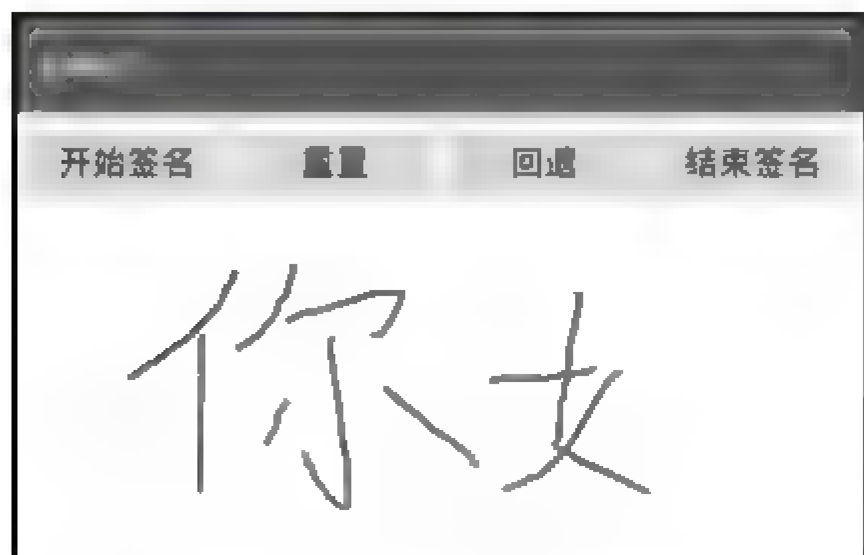


图 11-15　签名一半的界面

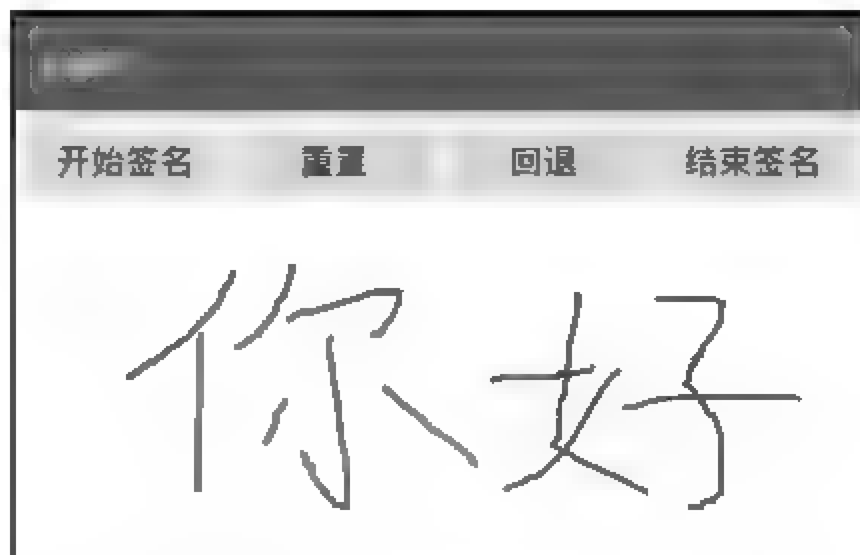


图 11-16　签名完成的界面

11.3　手势检测

本节介绍常见手势的检测与使用，首先说明手势检测器的原理与具体用法；然后阐述飞掠视图的基本用法，利用飞掠视图实现简单的横幅轮播；最后结合手势检测器与飞掠视图说明如何通过手势检测器控制横幅轮播的翻页动作。

11.3.1　手势检测器 GestureDetector

由于触摸事件的检测与识别比较烦琐，因此 Android 提供了手势检测器 GestureDetector 帮助开发者识别手势。利用 GestureDetector 可以自动辨别常用的几个手势事件，如点击、长按、滑动等，从而使开发者专注于业务逻辑，不必在手势的行为判断上绞尽脑汁。

下面是 GestureDetector 的常用方法。

- 构造函数：注册手势监听器 OnGestureListener，该监听器提供了若干种手势方法，需要重写以接管对应的事件处理。手势方法说明如下：
  - onDown：在用户按下时触发。
  - onShowPress：已按下但还未滑动或松开时触发，通常用于按下状态时的高亮显示。
  - onSingleTapUp：在用户轻点一下弹起时触发，通常用于点击事件。按下时间在 0.5 秒内为点击。
  - onScroll：在用户滑动过程中触发。
  - onLongPress：在用户长按时触发，通常用于长按事件。按下时间超过 0.5 秒为长按。
  - onFling：在用户飞快地滑出一段距离时触发，通常用于翻页事件。该方法的前两个参数为滑动开始和结束时的事件信息，后面两个参数分别为滑动操作在横坐标上的滑动速率和在纵坐标上的滑动速率。

  上述手势方法有部分需要返回布尔值，返回 true 表示该手势已经被处理了，其他人不需要再做无用功；返回 false 表示该手势没被处理，留给其他人处理。
- onTouchEvent：由手势检测器接管对应视图的触摸事件。

下面是使用 OnGestureListener 的代码：

```
public class GestureDetectorActivity extends AppCompatActivity {
    private TextView tv_gesture;
    private GestureDetector mGesture;
    private String desc = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_gesture_detector);
        tv_gesture = (TextView) findViewById(R.id.tv_gesture);
        mGesture = new GestureDetector(this, new MyGestureListener());
    }

    public boolean dispatchTouchEvent(MotionEvent event) {
        mGesture.onTouchEvent(event);
        return true;
    }

    final class MyGestureListener implements GestureDetector.OnGestureListener {
        @Override
        public final boolean onDown(MotionEvent event) {
            return true;  //onDown 的返回值没有作用，不影响其他手势的处理
        }

        @Override
        public final boolean onFling(MotionEvent e1、MotionEvent e2, float velocityX, float velocityY) {
            float offsetX = e1.getX() - e2.getX();
            float offsetY = e1.getY() - e2.getY();
            if (Math.abs(offsetX) > Math.abs(offsetY)) {
                if (offsetX > 0) {
                    desc = String.format("%s%s 您向左滑动了一下\n", desc, DateUtil.getNowTime());
                } else {
                    desc = String.format("%s%s 您向右滑动了一下\n", desc, DateUtil.getNowTime());
                }
            } else {
                if (offsetY > 0) {
                    desc = String.format("%s%s 您向上滑动了一下\n", desc, DateUtil.getNowTime());
                } else {
                    desc = String.format("%s%s 您向下滑动了一下\n", desc, DateUtil.getNowTime());
                }
            }
```

```
            tv_gesture.setText(desc);
            return true;
        }

        @Override
        public final void onLongPress(MotionEvent event) {
            desc = String.format("%s%s 您长按了一下 下\n", desc, DateUtil.getNowTime());
            tv_gesture.setText(desc);
        }

        @Override
        public final boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
            return false;
        }

        @Override
        public final void onShowPress(MotionEvent event) {
        }

        @Override
        public boolean onSingleTapUp(MotionEvent event) {
            desc = String.format("%s%s 您轻轻点了一下\n", desc, DateUtil.getNowTime());
            tv_gesture.setText(desc);
            return true;   //返回 true 表示已经处理了，其他地方不要再处理这个手势
        }
    }
}
```

手势检测器的使用效果如图 11-17 所示，可以发现检测到的手势包括点击、长按、上下左右滑动等。

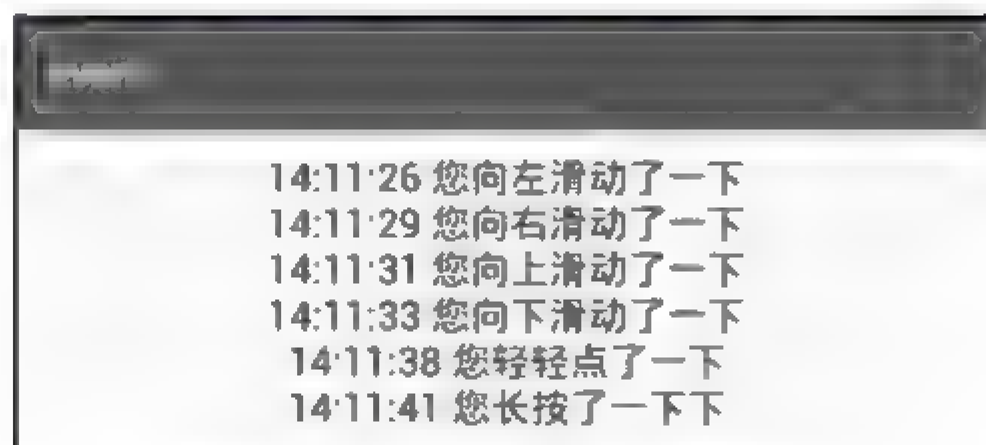


图 11-17　手势检测器的检测结果

11.3.2　飞掠视图 ViewFlipper

手机屏幕尺寸不大，为了在有限空间中展示尽可能多的信息，Android 设计了多种方式显示超出屏幕尺寸的界面，包括上下滚动、左右滑动等。飞掠视图 ViewFlipper 的层次翻动就是其中一项技术。与 ViewPager 相比，两者都是一系列类似视图的组合，ViewFlipper 更像是视

图的立体排列（如现实生活中的书籍），从上往下翻页；ViewPager更像是一幅长长的平面画卷，从左往右翻页。

下面是ViewFlipper的常用方法。

- setFlipInterval：设置每次翻页的时间间隔。单位毫秒。
- setAutoStart：设置是否自动开始翻页。为true表示自动开始。
- startFlipping：开始翻页。
- stopFlipping：停止翻页。
- isFlipping：判断当前是否正在翻页。
- showNext：显示下一个视图。
- showPrevious：显示上一个视图。
- setDisplayedChild：设置当前展示第几个视图。
- getDisplayedChild：获取当前展示的是第几个视图。
- setInAnimation：设置视图的移入动画。
- getInAnimation：获取移入动画的动画对象。
- setOutAnimation：设置视图的移出动画。
- getOutAnimation：获取移出动画的动画对象。

下面是利用ViewFlipper实现简单横幅轮播的代码：

```
public class ViewFlipperActivity extends AppCompatActivity implements OnClickListener {
    private Button btn_control_flipper;
    private RelativeLayout rl_content;
    private ViewFlipper vf_content;
    private RadioGroup rg_indicator;
    private int dip_15;
    private boolean bPlay = true;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_view_flipper);
        btn_control_flipper = (Button) findViewById(R.id.btn_control_flipper);
        rl_content = (RelativeLayout) findViewById(R.id.rl_content);
        vf_content = (ViewFlipper) findViewById(R.id.vf_content);
        rg_indicator = (RadioGroup) findViewById(R.id.rg_indicator);
        btn_control_flipper.setOnClickListener(this);
        findViewById(R.id.btn_pre_flipper).setOnClickListener(this);
        findViewById(R.id.btn_next_flipper).setOnClickListener(this);
        initFlipper();
    }

    private void initFlipper() {
```

```
        LayoutParams params = (LayoutParams) rl_content.getLayoutParams();
        params.height = (int) (DisplayUtil.getSreenWidth(this) * 250f/ 640f);
        rl_content.setLayoutParams(params);
        ArrayList<Integer> imageList = new ArrayList<Integer>();
        imageList.add(Integer.valueOf(R.drawable.banner_1));
        imageList.add(Integer.valueOf(R.drawable.banner_2));
        imageList.add(Integer.valueOf(R.drawable.banner_3));
        imageList.add(Integer.valueOf(R.drawable.banner_4));
        imageList.add(Integer.valueOf(R.drawable.banner_5));
        for (int i = 0; i < imageList.size(); i++) {
            Integer imageID = ((Integer) imageList.get(i)).intValue();
            ImageView iv_item = new ImageView(this);
            iv_item.setLayoutParams(new LayoutParams(
                    LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
            iv_item.setScaleType(ImageView.ScaleType.FIT_XY);
            iv_item.setImageResource(imageID);
            vf_content.addView(iv_item);
        }
        dip_15 = Utils.dip2px(this, 15);
        for (int i = 0; i < imageList.size(); i++) {
            RadioButton radio = new RadioButton(this);
            radio.setLayoutParams(new RadioGroup.LayoutParams(dip_15, dip_15));
            radio.setGravity(Gravity.CENTER);
            radio.setButtonDrawable(R.drawable.indicator_selector);
            rg_indicator.addView(radio);
        }
        vf_content.setDisplayedChild(0);
        vf_content.setAutoStart(true);
        mHandler.postDelayed(mRefresh, 200);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_pre_flipper) {
            vf_content.showPrevious();
        } else if (v.getId() == R.id.btn_next_flipper) {
            vf_content.showNext();
        } else if (v.getId() == R.id.btn_control_flipper) {
            bPlay = !bPlay;
            if (bPlay == true) {
                vf_content.startFlipping();
                btn_control_flipper.setText("停止自动翻页");
            } else {
```

```
                    vf_content.stopFlipping();
                    btn_control_flipper.setText("开始自动翻页");
                }
            }
        }

        private Handler mHandler = new Handler();
        private Runnable mRefresh = new Runnable() {
            @Override
            public void run() {
                int pos = vf_content.getDisplayedChild();
                ((RadioButton) rg_indicator.getChildAt(pos)).setChecked(true);
                mHandler.postDelayed(this, 200);
            }
        };
}
```

简单横幅轮播的效果如图11-18和图11-19所示。其中，图11-18所示为开始轮播的界面，通过按钮控制翻到上一页、翻到下一页或自动进行翻页；图11-19所示为轮播到第4张图片时的界面，轮播间隔既可以在代码中调用setFlipInterval方法设置，又可以直接在布局文件中指定flipInterval属性。

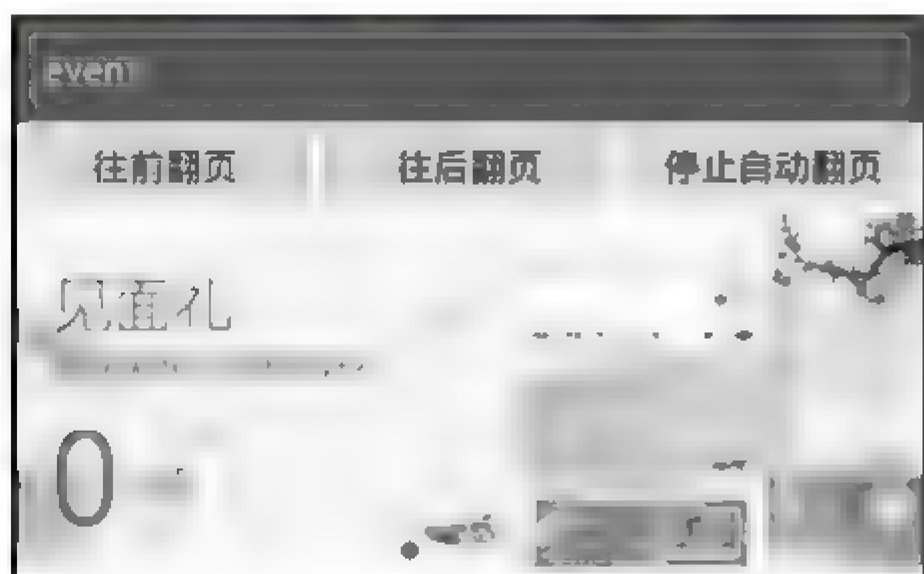


图11-18　飞掠视图开始轮播



图11-19　飞掠视图轮播到第4张

11.3.3　手势控制横幅轮播

前面演示简单横幅轮播时，需要通过按钮控制轮播动作，非常不便。接下来我们尝试结合手势检测器与飞掠视图实现手势控制的轮播效果。具体处理步骤如下：

步骤01 定义一个手势检测器的对象，并在自定义视图的dispatchTouchEvent方法中声明本视图的触摸事件由该检测器接管。

步骤02 实现手势监听器的onFling方法，在该方法内部判断播放上一页还是播放下一页，简单实现只需判断滑动前后的横坐标偏移是否超出阈值；若想更精确地校验，则可增加检查横坐标上的滑动速率是否达标，即判断velocityX是否超出阈值。

步骤03 做一个简单定时器，通过获取当前正在播放的视图编号设置下方指示器对应次序的高亮圆点。

下面是手势控制横幅轮播的自定义布局代码：

```
public class BannerFlipper extends RelativeLayout {
    private static final String TAG = "BannerFlipper";
    private Context mContext;
    private ViewFlipper mFlipper;
    private RadioGroup mGroup;
    private LayoutInflater mInflater;
    private int dip_15;
    private GestureDetector mGesture;
    private float mFlipGap = 20f;

    public BannerFlipper(Context context) {
        this(context, null);
    }

    public BannerFlipper(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        init();
    }

    public void setImage(ArrayList<Integer> imageList) {
        for (int i = 0; i < imageList.size(); i++) {
            Integer imageID = ((Integer) imageList.get(i)).intValue();
            ImageView iv_item = new ImageView(mContext);
            iv_item.setLayoutParams(new LayoutParams(
                    LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
            iv_item.setScaleType(ImageView.ScaleType.FIT_XY);
            iv_item.setImageResource(imageID);
            mFlipper.addView(iv_item);
        }
        for (int i = 0; i < imageList.size(); i++) {
            RadioButton radio = new RadioButton(mContext);
            radio.setLayoutParams(new RadioGroup.LayoutParams(dip_15, dip_15));
            radio.setGravity(Gravity.CENTER);
            radio.setButtonDrawable(R.drawable.indicator_selector);
            mGroup.addView(radio);
        }
        mFlipper.setDisplayedChild(imageList.size() - 1);
        startFlip();
    }
```



```
private void init() {
    mInflater = ((Activity) mContext).getLayoutInflater();
    View view = mInflater.inflate(R.layout.banner_flipper, null);
    mFlipper = (ViewFlipper) view.findViewById(R.id.banner_flipper);
    mGroup = (RadioGroup) view.findViewById(R.id.rg_indicator);
    addView(view);
    dip_15 = Utils.dip2px(mContext, 15);
    // 该手势的 onSingleTapUp 事件是点击时进入广告页
    mGesture = new GestureDetector(mContext, new BannerGestureListener());
    mHandler.postDelayed(mRefresh, 200);
}

public boolean dispatchTouchEvent(MotionEvent event) {
    mGesture.onTouchEvent(event);
    return true;
}

final class BannerGestureListener implements GestureDetector.OnGestureListener {
    @Override
    public final boolean onDown(MotionEvent event) {
        return true;
    }

    @Override
    public final boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
        if (e1.getX() - e2.getX() > mFlipGap) {
            startFlip();
            return true;
        }
        if (e1.getX() - e2.getX() < -mFlipGap) {
            backFlip();
            return true;
        }
        return false;
    }

    @Override
    public final void onLongPress(MotionEvent event) {
    }

    @Override
    public final boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
        return false;
```

```
            }

            @Override
            public final void onShowPress(MotionEvent event) {
            }

            @Override
            public boolean onSingleTapUp(MotionEvent event) {
                int position = mFlipper.getDisplayedChild();
                mListener.onBannerClick(position);
                return true;
            }
        }

        private void startFlip() {
            mFlipper.startFlipping();
            mFlipper.showNext();
        }

        private void backFlip() {
            mFlipper.startFlipping();
            mFlipper.showPrevious();
        }

        private Handler mHandler = new Handler();
        private Runnable mRefresh = new Runnable() {
            @Override
            public void run() {
                int pos = mFlipper.getDisplayedChild();
                ((RadioButton) mGroup.getChildAt(pos)).setChecked(true);
                mHandler.postDelayed(this, 200);
            }
        };

        private BannerClickListener mListener;
        public void setOnBannerListener(BannerClickListener listener) {
            mListener = listener;
        }

        public static interface BannerClickListener {
            public abstract void onBannerClick(int position);
        }
    }
```

手势控制横幅轮播的效果如图 11-20 和图 11-21 所示。其中，图 11-20 所示为开始轮播的界面，这里没有任何按钮，完全依靠手势的滑动控制左翻还是右翻；图 11-21 所示为手势从右往左滑过后的界面，从右往左滑表示翻到下一页，所以当前界面由第二页跳到第三页。



图 11-20　手势横幅开始轮播



图 11-21　手势滑动翻到下一页

11.4　手势冲突处理

本节介绍手势冲突的常见处理办法，对于上下滚动与左右滑动的冲突，既可由上级视图主动判断是否拦截，又可由下级视图根据情况向上级反馈是否允许拦截；对于内部滑动与翻页滑动的冲突，可以通过限定某块区域接管特定的手势实现对不同手势的区分处理。

11.4.1　上下滚动与左右滑动的冲突处理

Android 控件繁多，允许滚动或滑动操作的视图也不少，比如滚动视图 ScrollView、翻页视图 ViewPager 等，如果开发者要自己接管手势处理，像上一节手势控制横幅轮播那样处理，这个页面的滑动就存在重叠的情况，即很可能造成滑动冲突，系统响应了 A 视图的滑动事件，就顾不上 B 视图的滑动事件。

举个例子，某电商 App 的主页很长，内部采用滚动视图 ScrollView，允许上下滚动。该页面中央有一个手势控制的横幅轮播，如图 11-22 所示。用户在 Banner 上左右滑动，试图查看 Banner 的前后广告，结果如图 11-23 所示，翻页不成功，整个页面反而往上滚动了。

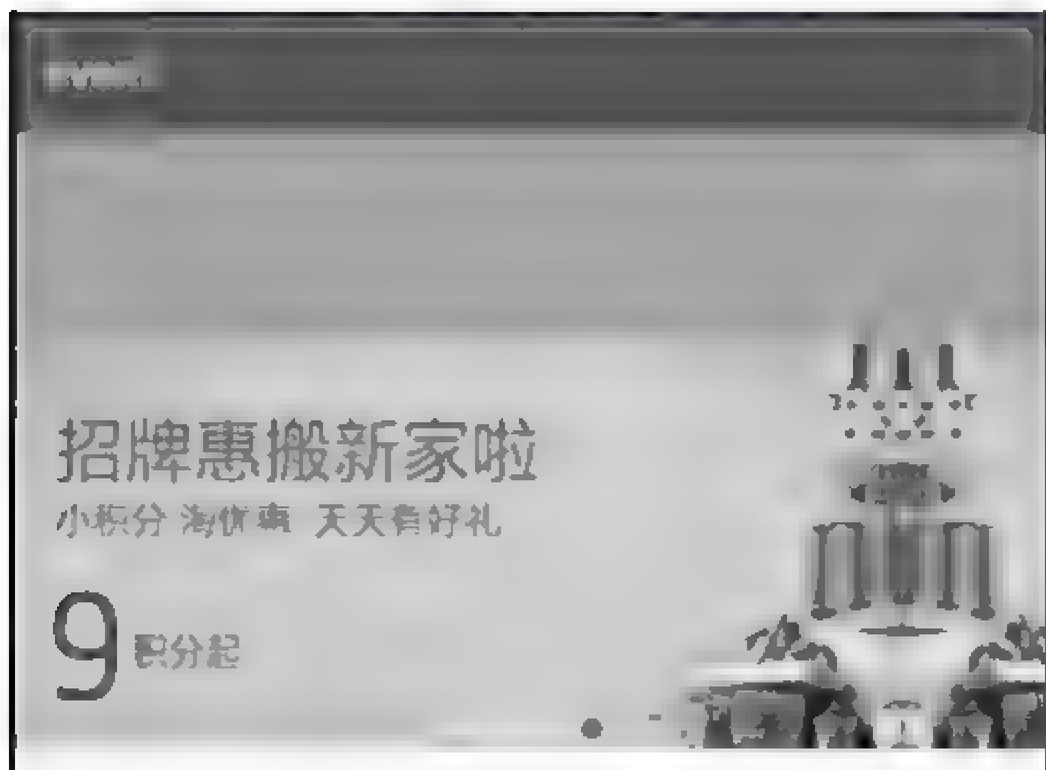


图 11-22　滚动视图中的横幅轮播



图 11-23　翻页滑动导致上下滚动

即使多次重复试验，仍然会发现 Banner 很少跟着翻页，而是继续上下滚动。因为 Banner 外层被 ScrollView 包着，系统检测到用户手势的一撇，上级领导 ScrollView 自作主张地认为用户要把页面往上拉，于是页面往上滚动，完全没考虑这一撇其实是用户想翻动 Banner。但是 ScrollView 不会考虑这些，因为没有告诉它超过多大斜率才可以上下滚动；既然没有通知，ScrollView 只要发现手势事件前后的纵坐标发生变化，就会一律进行上下滚动处理。

要解决这个滑动冲突，关键在于提供某种方式通知 ScrollView，告诉它什么时候可以上下滚动，什么时候不能上下滚动。这个通知方式主要有两种，一种是上级主动下乡体察民情，即由滚动视图判断滚动规则并决定是否拦截手势；另一种是下级向上反映民意，即由下级视图告诉滚动视图是否拦截手势。下面分别介绍这两种处理方式。

1. 由滚动视图判断滚动规则

前两节提到，容器类视图可以重写 onInterceptTouchEvent 方法，根据条件判断结果决定是否拦截发给下级的手势。我们可以自定义一个滚动视图，在 onInterceptTouchEvent 方法中判断本次手势的横坐标与纵坐标，如果纵坐标的偏移大于横坐标的偏移，此时就是垂直滚动，应拦截手势并交给自身进行上下滚动；否则表示此时为水平滚动，不应拦截手势，而是让下级视图处理左右滑动事件。

下面的代码用于演示自定义滚动视图拦截垂直滚动、同时放过水平滚动的功能。

```
public class CustomScrollView extends ScrollView {
    private float mOffsetX, mOffsetY;
    private float mLastPosX, mLastPosY;

    public CustomScrollView(Context context) {
        this(context, null);
    }

    public CustomScrollView(Context context, AttributeSet attr) {
        super(context, attr);
    }

    @Override
    public boolean onInterceptTouchEvent(MotionEvent event) {
        boolean result = false;
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            mOffsetX = 0.0F;
            mOffsetY = 0.0F;
            mLastPosX = event.getX();
            mLastPosY = event.getY();
            result = super.onInterceptTouchEvent(event); // false 传给子控件
            break;
        default:
```

```
                float thisPosX = event.getX();
                float thisPosY = event.getY();
                mOffsetX += Math.abs(thisPosX - mLastPosX); // x 轴偏差
                mOffsetY += Math.abs(thisPosY - mLastPosY); // y 轴偏差
                mLastPosX = thisPosX;
                mLastPosY = thisPosY;
                if (mOffsetX < 3 && mOffsetY < 3) {
                    result = false; // false 传给子控件（点击事件）
                } else if (mOffsetX < mOffsetY) {
                    result = true; // true 不传给子控件（垂直滑动）
                } else {
                    result = false; // false 传给子控件
                }
                break;
        }
        return result;
    }
}
```

接着在XML布局文件中把ScrollView节点改为自定义滚动视图的完整路径名称（如com.example.event.widget.CustomScrollView），重新运行App后查看横幅轮播，手势滑动效果如图11-24所示。此时翻页成功，且整个页面固定不动，未发生上下滚动的情况。



图 11-24　翻页滑动未造成上下滚动

2. 下级视图告诉滚动视图能否拦截手势

目前的案例中，ScrollView下面只有Banner一个淘气鬼，所以允许单独给它开小灶。在实际场合中，往往有多个调皮鬼，一个要吃苹果，另一个要吃香蕉，倘若都要ScrollView帮忙，那可真是众口难调，忙都忙不过来了。不如弄个水果篮，让这些小屁孩自己去拿，要吃苹果的就拿苹果，要吃香蕉的就拿香蕉，如此皆大欢喜，再也不用大人劳心劳力了。

具体到代码的实现，是调用requestDisallowInterceptTouchEvent方法，该方法的参数为true时，表示禁止上级拦截触摸事件。至于何时调用该方法，当然是在检测到滑动前后的横坐标偏移大于纵坐标偏移了。对于Banner采用手势监听器的情况，可重写监听器的onScroll方法，在该方法中加入坐标偏移的判断，代码如下：

```
        public final boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY){
            // 如果外层是普通的ScrollView，此处就不允许父容器的拦截动作
            if (Math.abs(distanceY) < Math.abs(distanceX)) {
                BannerFlipper.this.getParent().requestDisallowInterceptTouchEvent(true);
                return true;
```

```
            } else {
                return false;
            }
        }
```

修改后的手势滑动效果如图 11-24 所示。左右滑动能够正常翻页，整个页面也不容易上下滚动了。

11.4.2 内部滑动与翻页滑动的冲突处理

在前面的手势冲突中，ScrollView 是上级视图，有时也是下级视图，比如页面采用 ViewPager 布局，每个 Fragment 之间是左右滑动的关系，每个 Fragment 都可以拥有自己的 ScrollView。如此一来，在左右滑动时，ScrollView 反而变成 ViewPager 的下级，这样前面的冲突处理办法不能奏效了，只能另想办法。

自定义一个基于 ViewPager 的翻页视图是一种思路；另一种思路可借鉴 Android 自带的抽屉布局 DrawerLayout，该布局视图允许左右滑动，在滑动时会拉出侧面的抽屉面板，常用于实现侧滑菜单。抽屉布局与翻页视图在滑动方面有区别，翻页视图在内部的任何位置均可触发滑动事件，而抽屉布局只在屏幕两侧边缘才会触发滑动事件。

举个实际应用的例子，微信的聊天窗口是上下滚动的，在主窗口的大部分区域触摸都是上下滚动窗口，若在窗口左侧边缘按下再右拉，则可看到左边拉出了消息关注页面。限定某块区域接管特定的手势是处理滑动冲突的另一种行之有效的方法。

既然提到了抽屉布局，不妨稍微了解一下。下面是 DrawerLayout 的常用方法说明。

- setDrawerShadow：设置主页面的渐变阴影图形。
- addDrawerListener：添加抽屉面板的拉出监听器。需实现监听器 DrawerListener 的 4 个方法。
    - onDrawerSlide：抽屉面板滑动时触发。
    - onDrawerOpened：抽屉面板打开时触发。
    - onDrawerClosed：抽屉面板关闭时触发。
    - onDrawerStateChanged：抽屉面板的状态发生变化时触发。
- removeDrawerListener：移除抽屉面板的拉出监听器。
- closeDrawers：关闭所有抽屉面板。
- openDrawer：打开指定抽屉面板。
- closeDrawer：关闭指定抽屉面板。
- isDrawerOpen：判断指定抽屉面板是否打开。

抽屉布局不但可以拉出左侧抽屉面板，而且可以拉出右侧抽屉面板。左侧面板与右侧面板的区别在于：左侧面板在布局文件中的 layout_gravity 属性为 left，而右侧面板在布局文件中的 layout_gravity 属性为 right。

下面是使用 DrawerLayout 的布局文件：



```
<android.support.v4.widget.DrawerLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/dl layout"
    android:layout width="match parent"
    android:layout height="match parent" >

    <LinearLayout
        android:layout width="match parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >

            <Button
                android:id="@+id/btn_drawer_left"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:gravity="center"
                android:text="打开左边侧滑"
                android:textColor="@color/black"
                android:textSize="20sp" />

            <Button
                android:id="@+id/btn_drawer_right"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:gravity="center"
                android:text="打开右边侧滑"
                android:textColor="@color/black"
                android:textSize="20sp" />
        </LinearLayout>

        <TextView
            android:id="@+id/tv_drawer_center"
            android:layout_width="match parent"
            android:layout_height="0dp"
            android:layout weight="1"
            android:gravity="top|center"
            android:paddingTop="30dp"
```

```
            android:text="这里是首页"
            android:textColor="@color/black"
            android:textSize="20sp" />
    </LinearLayout>

    <ListView
        android:id="@+id/lv_drawer_left"
        android:layout_width="150dp"
        android:layout_height="match_parent"
        android:layout_gravity="left"
        android:background="#ffdd99" />

    <ListView
        android:id="@+id/lv_drawer_right"
        android:layout_width="150dp"
        android:layout_height="match_parent"
        android:layout_gravity="right"
        android:background="#99ffdd" />
</android.support.v4.widget.DrawerLayout>
```

上述布局文件对应的页面代码如下：

```
public class DrawerLayoutActivity extends AppCompatActivity implements OnClickListener {
    private final static String TAG = "DrawerLayoutActivity";
    private DrawerLayout dl_layout;
    private Button btn_drawer_left;
    private Button btn_drawer_right;
    private TextView tv_drawer_center;
    private ListView lv_drawer_left;
    private ListView lv_drawer_right;
    private String[] titleArray = { "首页", "新闻", "娱乐", "博客", "论坛" };
    private String[] settingArray = { "我的", "设置", "关于" };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drawer_layout);
        dl_layout = (DrawerLayout) findViewById(R.id.dl_layout);
        dl_layout.addDrawerListener(new SlidingListener());
        btn_drawer_left = (Button) findViewById(R.id.btn_drawer_left);
        btn_drawer_right = (Button) findViewById(R.id.btn_drawer_right);
        tv_drawer_center = (TextView) findViewById(R.id.tv_drawer_center);
        btn_drawer_left.setOnClickListener(this);
        btn_drawer_right.setOnClickListener(this);
```

```
        lv_drawer_left = (ListView) findViewById(R.id.lv_drawer_left);
        ArrayAdapter<String> left_adapter = new ArrayAdapter<String>(this,
                R.layout.item_select, titleArray);
        lv_drawer_left.setAdapter(left_adapter);
        lv_drawer_left.setOnItemClickListener(new LeftListListener());
        lv_drawer_right = (ListView) findViewById(R.id.lv_drawer_right);
        ArrayAdapter<String> right_adapter = new ArrayAdapter<String>(this,
                R.layout.item_select, settingArray);
        lv_drawer_right.setAdapter(right_adapter);
        lv_drawer_right.setOnItemClickListener(new RightListListener());
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_drawer_left) {
            if (dl_layout.isDrawerOpen(lv_drawer_left)) {
                dl_layout.closeDrawer(lv_drawer_left);
            } else {
                dl_layout.openDrawer(lv_drawer_left);
            }
        } else if (v.getId() == R.id.btn_drawer_right) {
            if (dl_layout.isDrawerOpen(lv_drawer_right)) {
                dl_layout.closeDrawer(lv_drawer_right);
            } else {
                dl_layout.openDrawer(lv_drawer_right);
            }
        }
    }

    private class LeftListListener implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            String text = titleArray[position];
            tv_drawer_center.setText("这里是" + text + "页面");
            dl_layout.closeDrawers();
        }
    }

    private class RightListListener implements OnItemClickListener {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            String text = settingArray[position];
```

```
                tv_drawer_center.setText("这里是" + text + "页面");
                dl_layout.closeDrawers();
            }
        }

    private class SlidingListener implements DrawerListener {
        @Override
        public void onDrawerSlide(View paramView, float paramFloat) {
        }

        @Override
        public void onDrawerOpened(View paramView) {
            if (paramView.getId() == R.id.lv_drawer_left) {
                btn_drawer_left.setText("关闭左边侧滑");
            } else {
                btn_drawer_right.setText("关闭右边侧滑");
            }
        }

        @Override
        public void onDrawerClosed(View paramView) {
            if (paramView.getId() == R.id.lv_drawer_left) {
                btn_drawer_left.setText("打开左边侧滑");
            } else {
                btn_drawer_right.setText("打开右边侧滑");
            }
        }

        @Override
        public void onDrawerStateChanged(int paramInt) {
        }
    }
}
```

抽屉布局的展示效果如图 11-25、图 11-26、图 11-27 所示。其中，图 11-25 所示为初始页面，图 11-26 所示为在左侧边缘拉出左边侧滑菜单的界面，图 11-27 所示为在右侧边缘拉出右边侧滑菜单的界面。

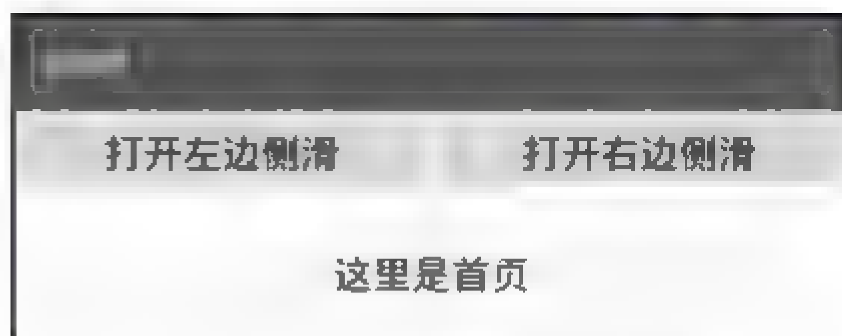


图 11-25 演示抽屉布局的的初始界面

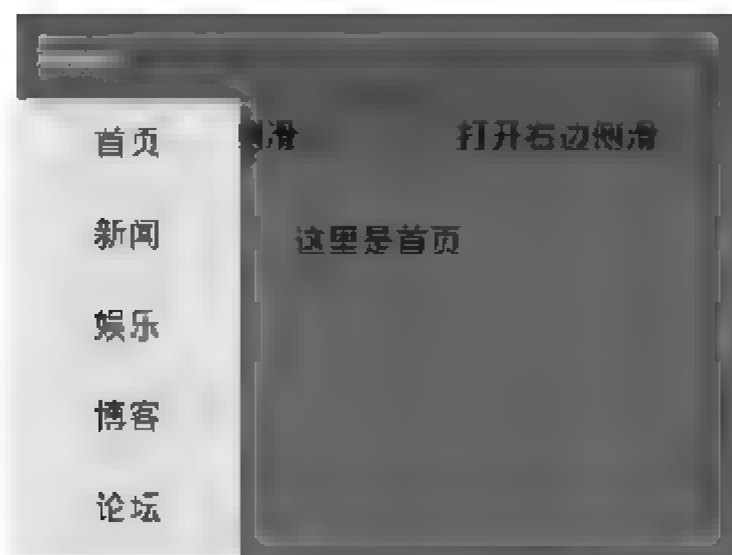


图 11-26　左侧边缘拉出侧滑菜单

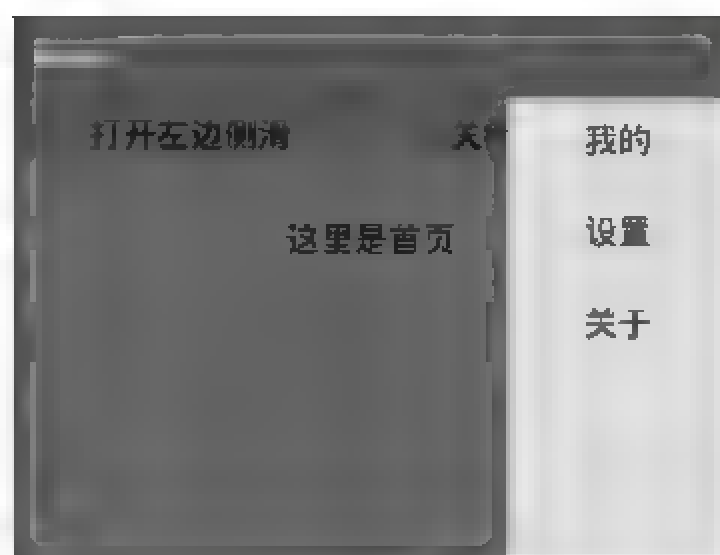


图 11-27　右侧边缘拉出侧滑菜单

11.5　实战项目：抠图神器——美图变变

程序员通常是闷骚的宅男，对技术的钻研孜孜不倦，不过一味地追求技术深度，不见得就能登上巅峰。譬如智能手机行业，以技术制胜的华为和小米，销量反而不敌 OPPO 和 vivo。究其原因，多半是后者认真对待用户需求，从用户体验的痛点下手，推出了自拍美颜等手机，由此收获了大批客户。本章的实战项目不求技术有多广、多深，只求有没有用、好不好用。所谓抠图神器，就是从一幅图片中抠出用户想要的某块区域。就像在花店里卖花，先适当修剪花束，再配上一些包装，顿时看起来美美哒，不愁用户不喜欢。

11.5.1　设计思路

这里说的美图变变，其实就是一个抠图工具，通过对图像进行平移、缩放、旋转等操作，把图像的某个区域抠下来。图 11-28 所示为美图变变的效果图，中间高亮部分为待抠区域，西湖后面的雷峰塔太小了，现在准备把雷峰塔先拉近再放大，然后抠出来。



图 11-28　美图变变的抠图效果

这个效果图的界面很简洁，主界面没有任何控制按钮，完全靠手势操作。实现的手势处理有以下 6 种。

- 长按手势：在页面任何一处长按 0.5 秒以上，即可触发长按事件，弹出文件菜单后选择打开图片或保存图片。
- 移动高亮区域的手势：点击高亮区域内部，再滑动手势，即可将高亮区域拖曳至指定位置。
- 调整高亮区域边界的手势：点击高亮区域边界，再滑动手势，即可将边界拉动至指定位置。
- 移动图片的手势：点击高亮区域外部（阴影部分），然后滑动手势，即可将整张图片拖曳至指定位置。
- 缩放图片的手势：两只手指同时按压屏幕，然后一起往中心点接近或远离，即可实现图片的缩小和放大操作。

- 旋转图片的手势：两只手指同时按压屏幕，然后围绕中心点一起顺时针或逆时针转动，即可实现图片的旋转操作。

长按和移动手势的判断相对简单，根据按压时长或按压坐标就能判断属于哪类手势。缩放与旋转手势的判断相对复杂，涉及多点触控和三角函数相关知识，主要思路是：记录两只手指移动前的坐标和移动后的坐标，总共 4 个坐标点，然后分别计算移动前的两指距离和移动后的两指距离，判断两个距离的差是否大于两指移动距离之和的二分之根号二倍。判断结果若是大于，则表示本次为缩放手势，否则为旋转手势，接着计算缩放比例或旋转角度即可。

11.5.2 小知识：图像的基本加工

Android 上的图形使用 Drawable 类，位图管理使用 Bitmap 类。Drawable 用于在界面上展示图片，Bitmap 用于对图像数据进行加工操作，图像加工操作包括平移、缩放、旋转、裁剪等。这两个类之间的转换通过 BitmapDrawable 完成。

其中，Bitmap 转 Drawable 的代码如下：

```
Drawable drawable = new BitmapDrawable(getResources(), bitmap);
```

Drawable 转 Bitmap 的代码如下：

```
Bitmap bitmap = ((BitmapDrawable)drawable).getBitmap();
```

下面是 Bitmap 的常用方法说明。

- createBitmap：从源图像中裁剪一块位图区域。
- createScaledBitmap：根据设定的图片大小从源图像获得缩放后的新图像。
- compress：根据设定的位图格式与压缩质量对图像进行压缩。
- recycle：回收位图对象资源。
- getByteCount：获取位图对象的字节大小。
- getWidth：获取位图对象的宽度。
- getHeight：获取位图对象的高度。

了解这些方法的使用说明后，就可以实现图像的基本加工操作了。

（1）图像裁剪：调用 Bitmap 类的 createBitmap 方法时，指定裁剪图像的上、下、左、右边界即可。

（2）图像平移：调用 Canvas 对象的 drawBitmap 时，指定图像绘制的起始点位置即可。

（3）图像缩放：调用 Bitmap 类的 createScaledBitmap 方法时，指定新图像宽和高的数值即可。

（4）图像旋转：需要借助矩阵工具 Matrix，先调用 Matrix 对象的 postRotate 方法设置旋转角度，再根据设置好的矩阵对象调用 createBitmap 方法创建旋转图像，转换代码如下：

```
public static Bitmap getRotateBitmap(Bitmap b, float rotateDegree) {
        Matrix matrix = new Matrix();
        matrix.postRotate((float) rotateDegree);
```

```
        Bitmap rotaBitmap = Bitmap.createBitmap(b, 0, 0, b.getWidth(), b.getHeight(), matrix, false);
        return rotaBitmap;
    }
```

图像变换的效果如图 11-29、图 11-30、图 11-31 所示。其中，图 11-29 所示为原始的图像界面，图 11-30 所示为放大两倍后的图像界面，图 11-31 所示为顺时针旋转 90 度后的图像界面。



图 11-29　变换前的原始图像



图 11-30　放大两倍后的图像



图 11-31　旋转 90 度后的图像

11.5.3　代码示例

编码与测试方面需要注意以下 3 点：

（1）对图片文件进行打开和保存操作，记得为 AndroidManifest.xml 添加对应的权限配置。

```
<!-- SD 卡 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

（2）长按弹出文件菜单，需要在 res/menu 目录下添加菜单布局文件 menu_meitu.xml。

（3）要在真机上测试实战项目，因为模拟器不支持多点触控，只有真机才能测试手势的缩放与旋转操作。

测试时，首先在实战项目的界面上长按，弹出读取图片文件的菜单，如图 11-32 所示。

点击“打开图片”，打开待加工的图片文件，拖动原始图片与高亮区域，并适当放大与旋转图片，使雷峰塔位于高亮区域中上部。期间的界面效果如图 11-33 与图 11-34 所示。其中，图 11-33 所示为刚打开图片时的初始界面；图 11-34 所示为手势调整结束，准备完成抠图时的界面。

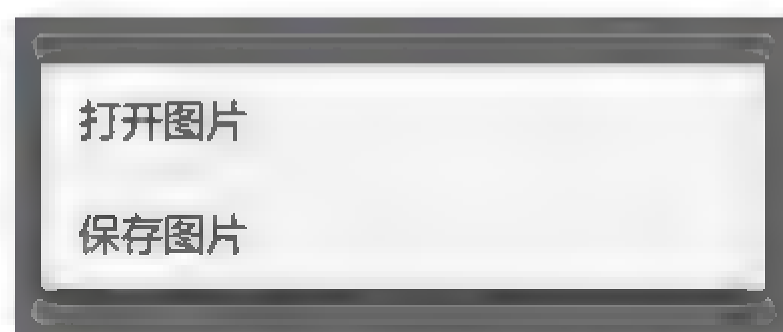


图 11-32　长按主页面弹出读写图片文件的菜单

接下来保存抠图完成的图片，在界面上长按，弹出读取图片文件的菜单，如图 11-35 所示。



图 11-33　抠图开始前的界面



图 11-34　抠图完成后的界面



图 11-35　长按主页面准备保存抠图

点击“保存图片”，填写保存后的文件名，完成图片保存操作，即可在指定的路径找到抠下来的图片。

下面是响应抠图手势自定义视图的代码：

```
public class MeituView extends View {
    private Context mContext;
    private Paint mPaintShade;

    public MeituView(Context context) {
        this(context, null);
    }

    public MeituView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        mPaintShade = new Paint();
        mPaintShade.setStyle(Style.FILL);
        mPaintShade.setColor(0x99000000);
    }

    private Bitmap mOrigBitmap = null;
    private Bitmap mCropBitmap = null;
    private Rect mRect = new Rect(0,0,0,0);
    public void setOrigBitmap(Bitmap orig) {
        mOrigBitmap = orig;
    }

    public Bitmap getCropBitmap() {
        return mCropBitmap;
    }

    public boolean setBitmapRect(Rect rect) {
```



```
        if (mOrigBitmap == null) {
            return false;
        } else if (rect.left<0 || rect.left>mOrigBitmap.getWidth()) {
            return false;
        } else if (rect.top<0 || rect.top>mOrigBitmap.getHeight()) {
            return false;
        } else if (rect.right<=0 || rect.left+rect.right>mOrigBitmap.getWidth()) {
            return false;
        } else if (rect.bottom<=0 || rect.top+rect.bottom>mOrigBitmap.getHeight()) {
            return false;
        }
        mRect = rect;
        mCropBitmap = Bitmap.createBitmap(mOrigBitmap,
                mRect.left, mRect.top, mRect.right, mRect.bottom);
        postInvalidate();
        return true;
    }

    public Rect getBitmapRect() {
        return mRect;
    }

    private boolean bReset = false;
    private float mLastOffsetX, mLastOffsetY;
    private float mLastOffsetXTwo, mLastOffsetYTwo;
    private long mOriginTime;

    @Override
    protected void dispatchDraw(Canvas canvas) {
        super.dispatchDraw(canvas);
        if (mOrigBitmap == null) {
            return;
        }
        Rect rectShade = new Rect(0, 0, getMeasuredWidth(), getMeasuredHeight());
        canvas.drawRect(rectShade, mPaintShade);    // 画外圈阴影
        canvas.drawBitmap(mCropBitmap, mRect.left, mRect.top, new Paint());    // 画高亮处的图像
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int action = event.getAction() & MotionEvent.ACTION_MASK;
        switch (action) {
        case MotionEvent.ACTION_DOWN:
            mOriginTime = event.getEventTime();
            mOriginX = event.getX();
```

```
            mOriginY = event.getY();
            mOriginRect = mRect;
            mDragMode = getDragMode(mOriginX, mOriginY);
            bReset = true;
            break;
        case MotionEvent.ACTION_UP:
            if (mListener != null && Math.abs(event.getX()-mOriginX)<10 &&
                        Math.abs(event.getY()-mOriginY)<10) {
                if (event.getEventTime() - mOriginTime < 500) {   // 判断点击还是长按
                    mListener.onImageClick();
                } else {
                    mListener.onImageLongClick();
                }
            }
            break;
        case MotionEvent.ACTION_POINTER_DOWN:
            mDragMode = IMAGE_SCALE_OR_ROTATE;   // 另需判断缩放还是旋转
            bReset = true;
            break;
        case MotionEvent.ACTION_POINTER_UP:
            mDragMode = DRAG_NONE;
            break;
        case MotionEvent.ACTION_MOVE:
            int offsetX = (int) (event.getX()-mOriginX);
            int offsetY = (int) (event.getY()-mOriginY);
            Rect rect = null;
            int left = mOriginRect.left;
            int top = mOriginRect.top;
            int right = mOriginRect.right;
            int bottom = mOriginRect.bottom;
            if (mDragMode == DRAG_NONE) {
                return true;
            } else if (mDragMode == DRAG_WHOLE) {
                rect = new Rect(left+offsetX, top+offsetY, right, bottom);
            } else if (mDragMode == DRAG_LEFT) {
                rect = new Rect(left+offsetX, top, right-offsetX, bottom);
            } else if (mDragMode == DRAG_RIGHT) {
                rect = new Rect(left, top, right+offsetX, bottom);
            } else if (mDragMode == DRAG_TOP) {
                rect = new Rect(left, top+offsetY, right, bottom-offsetY);
            } else if (mDragMode == DRAG_BOTTOM) {
                rect = new Rect(left, top, right, bottom+offsetY);
            } else if (mDragMode == DRAG_LEFT_TOP) {
```

```
                rect = new Rect(left+offsetX, top+offsetY, right-offsetX, bottom-offsetY);
            } else if (mDragMode == DRAG_RIGHT_TOP) {
                rect = new Rect(left, top+offsetY, right+offsetX, bottom-offsetY);
            } else if (mDragMode == DRAG_LEFT_BOTTOM) {
                rect = new Rect(left+offsetX, top, right-offsetX, bottom+offsetY);
            } else if (mDragMode == DRAG_RIGHT_BOTTOM) {
                rect = new Rect(left, top, right+offsetX, bottom+offsetY);
            } else if (mDragMode == IMAGE_TRANSLATE) {
                if (mListener != null) {
                    mListener.onImageTraslate(offsetX, offsetY, bReset);
                    bReset = false;
                }
            } else if (mDragMode == IMAGE_SCALE_OR_ROTATE) {
                if (mListener != null) {
                    float nowWholeDistance = distance(event.getX(), event.getY(),
                            event.getX(1), event.getY(1));
                    float preWholeDistance = distance(mLastOffsetX, mLastOffsetY,
                            mLastOffsetXTwo, mLastOffsetYTwo);
                    float primaryDistance = distance(event.getX(), event.getY(),
                            mLastOffsetX, mLastOffsetY);
                    float secondaryDistance = distance(event.getX(1), event.getY(1),
                            mLastOffsetXTwo, mLastOffsetYTwo);
                    if (Math.abs(nowWholeDistance-preWholeDistance) >
                        (float) Math.sqrt(2)/2.0f*(primaryDistance+secondaryDistance)) {   //缩放
                        mListener.onImageScale(nowWholeDistance / preWholeDistance);
                    } else {   // 旋转
                        int preDegree = degree(mLastOffsetX, mLastOffsetY,
                                mLastOffsetXTwo, mLastOffsetYTwo);
                        int nowDegree = degree(event.getX(), event.getY(),
                                event.getX(1), event.getY(1));
                        mListener.onImageRotate(nowDegree - preDegree);
                    }
                }
            }
            if (mDragMode!=IMAGE_TRANSLATE && mDragMode!=IMAGE_SCALE_OR_ROTATE) {
                setBitmapRect(rect);
            }
            break;
        default:
            break;
        }
        mLastOffsetX = event.getX();
        mLastOffsetY = event.getY();
```

```
        if (event.getPointerCount() >= 2) {
            mLastOffsetXTwo = event.getX(1);
            mLastOffsetYTwo = event.getY(1);
        }
        return true;
    }

    private float distance(float x1, float y1, float x2, float y2) {
        float offsetX = x2 - x1;
        float offsetY = y2 - y1;
        return (float) Math.sqrt(offsetX*offsetX + offsetY*offsetY);
    }

    private int degree(float x1, float y1, float x2, float y2) {
        return (int) (Math.atan((y2-y1) / (x2-x1)) / Math.PI * 180);
    }

    private int DRAG_NONE = 0, DRAG_WHOLE = 1, DRAG_LEFT = 2,   DRAG_RIGHT = 3;
    private int DRAG_TOP = 4, DRAG_BOTTOM = 5, DRAG_LEFT_TOP = 6, DRAG_RIGHT_TOP = 7;
    private int DRAG_LEFT_BOTTOM = 8, DRAG_RIGHT_BOTTOM = 9;
    private int IMAGE_TRANSLATE = 10, IMAGE_SCALE_OR_ROTATE = 11;
    private int mDragMode = DRAG_NONE;
    private int mInterval = 15;
    private float mOriginX, mOriginY;
    private Rect mOriginRect;

    private int getDragMode(float f, float g) {
        int left = mRect.left;
        int top = mRect.top;
        int right = mRect.left + mRect.right;
        int bottom = mRect.top + mRect.bottom;
        if (Math.abs(f-left)<=mInterval && Math.abs(g-top)<=mInterval) {
            return DRAG_LEFT_TOP;
        } else if (Math.abs(f-right)<=mInterval && Math.abs(g-top)<=mInterval) {
            return DRAG_RIGHT_TOP;
        } else if (Math.abs(f-left)<=mInterval && Math.abs(g-bottom)<=mInterval) {
            return DRAG_LEFT_BOTTOM;
        } else if (Math.abs(f-right)<=mInterval && Math.abs(g-bottom)<=mInterval) {
            return DRAG_RIGHT_BOTTOM;
        } else if (Math.abs(f-left)<=mInterval && g>top+mInterval && g<bottom-mInterval) {
            return DRAG_LEFT;
        } else if (Math.abs(f-right)<=mInterval && g>top+mInterval && g<bottom-mInterval) {
            return DRAG_RIGHT;
        } else if (Math.abs(f-left)<=mInterval && g>top+mInterval && g<bottom-mInterval) {
            return DRAG_LEFT;
```

```
            } else if (Math.abs(g-top)<=mInterval && f>left+mInterval && f<right-mInterval) {
                return DRAG_TOP;
            } else if (Math.abs(g-bottom)<=mInterval && f>left+mInterval && f<right-mInterval) {
                return DRAG_BOTTOM;
            } else if (f>left+mInterval && f<right-mInterval && g>top+mInterval && g<bottom-mInterval) {
                return DRAG_WHOLE;
            } else if (f+mInterval<left || f-mInterval>right || g+mInterval<top || g-mInterval>bottom) {
                return IMAGE_TRANSLATE;
            } else {
                return DRAG_NONE;
            }
        }

        private ImageChangetListener mListener;
        public void setImageChangetListener(ImageChangetListener listener) {
            mListener = listener;
        }

        public static interface ImageChangetListener {
            public abstract void onImageClick();
            public abstract void onImageLongClick();
            public abstract void onImageTraslate(int offsetX, int offsetY, boolean bReset);
            public abstract void onImageScale(float ratio);
            public abstract void onImageRotate(int degree);
        }
    }
```

11.6 小　结

本章主要介绍了App开发用到的常见事件处理，包括按键事件的检测与处理（检测软键盘、检测物理按键、音量调节对话框）、触摸事件的检测与处理（手势事件的分发流程、手势事件处理MotionEvent、手写签名）、手势检测的实现与用法（手势检测器、飞掠视图、手势控制横幅轮播）、手势冲突的处理方式（上下滚动与左右滑动的冲突处理、内部滑动与翻页滑动的冲突处理）。最后设计了一个实战项目“抠图神器——美图变变”，在该项目的App编码中采用了本章介绍的主要手势事件，包括单点触摸、多点触控等。另外，介绍了图像的基本加工操作。

通过本章的学习，读者应该能够掌握以下4种开发技能：

（1）学会在合适的场合监听并处理按键事件。

（2）学会检测触摸事件并接管手势处理。

（3）学会使用主要的手势检测手段。

（4）学会避免手势冲突的情况发生。





动　画

本章介绍 App 开发常见的动画显示技术，主要包括如何使用帧动画实现电影播放效果、如何使用补间动画完成视图的4种基本状态变化、如何使用属性动画实现视图各种状态的动态变换效果以及动画技术常用的3种代表手段。最后结合本章所学的知识演示一个实战项目“仿QQ空间的动感影集”的设计与实现。

12.1 帧　动　画

本节介绍帧动画相关的技术实现，首先说明如何通过动画图形与宿主视图播放帧动画，接着阐述播放 GIF 动画存在的问题和对应的解决思路与技术方案，最后介绍如何使用过渡图形实现两幅图片之间的淡入、淡出动画。

12.1.1 帧动画的实现

Android 的动画分为三大类：帧动画、补间动画和属性动画。其中，帧动画是实现原理最简单的一种，跟现实生活中的电影胶卷类似，都是在短时间内连续播放多张图片，从而模拟动态画面的效果。

具体到代码实现，帧动画由动画图形 AnimationDrawable 生成。下面是 AnimationDrawable 的常用方法。

- addFrame：添加一幅图片帧，并指定该帧的持续时间（单位毫秒）。
- setOneShot：设置是否只播放一次。为 true 表示只播放一次，为 false 表示循环播放。
- start：开始播放。注意，设置宿主视图后才能进行播放。
- stop：停止播放。
- isRunning：判断是否正在播放。

有了动画图形，还得有一个宿主视图显示该图形，一般使用图像视图 ImageView 承载 AnimationDrawable，即调用 ImageView 对象的 setImageDrawable 方法将动画图形加载到图像视图中。

下面是播放帧动画的代码片段：

```
private void showFrameAnimByCode() {
    //帧动画需要把每帧图片加入 AnimationDrawable 队列
    ad_frame = new AnimationDrawable();
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p1), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p2), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p3), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p4), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p5), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p6), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p7), 50);
    ad_frame.addFrame(getResources().getDrawable(R.drawable.flow_p8), 50);
    ad_frame.setOneShot(false);
    iv_frame_anim.setImageDrawable(ad_frame);
    ad_frame.start();
}
```

帧动画的播放效果如图 12-1、图 12-2、图 12-3 所示。这组帧动画实际由 8 张瀑布图片构成，图中所示的 3 张画面为其中的 3 个瀑布帧，单看画面区别不大，连起来播放才能看到瀑布的流水动画。



图 12-1 瀑布动画帧 1



图 12-2 瀑布动画帧 2



图 12-3 瀑布动画帧 3

除了在代码中添加帧图片，还可以先把帧图片的排列定义在一个 XML 文件中；然后在代码中直接调用 ImageView 对象的 setImageResource 方法，加载帧动画的图形定义文件；再调用 ImageView 对象的 getDrawable 方法，获得动画图形的实例，并进行后续的播放操作。

下面是帧图片的定义文件：

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android" android:oneshot="false" >
    <item android:drawable="@drawable/flow_p1" android:duration="50"/>
    <item android:drawable="@drawable/flow_p2" android:duration="50"/>
    <item android:drawable="@drawable/flow_p3" android:duration="50"/>
    <item android:drawable="@drawable/flow_p4" android:duration="50"/>
    <item android:drawable="@drawable/flow_p5" android:duration="50"/>
    <item android:drawable="@drawable/flow_p6" android:duration="50"/>
    <item android:drawable="@drawable/flow_p7" android:duration="50"/>
    <item android:drawable="@drawable/flow_p8" android:duration="50"/>
</animation-list>
```

从图形定义文件中播放帧动画的效果与在代码中添加帧图片是一样的，代码如下：

```
    private void showFrameAnimByXml() {
        iv_frame_anim.setImageResource(R.drawable.frame_anim);
        ad_frame = (AnimationDrawable) iv_frame_anim.getDrawable();
        ad_frame.start();
    }
```

12.1.2 显示 GIF 动画

GIF 在 Windows 上是常见的图片格式，主要用来播放短小的动画。Android 虽然号称支持 PNG、JPG、GIF 三种图片格式，但是并不支持直接播放 GIF 动图，如果在图像视图中加载一张 GIF 文件，只会显示 GIF 文件的第一帧图片。

要想在手机上显示 GIF 文件，就要借助于帧动画技术，具体的实现方式主要有以下两种：

（1）开发者在电脑上把 GIF 文件手工分解为一组帧图片，放入工程的资源目录中，再通过动画图形显示帧动画。

（2）在代码中将 GIF 文件自动分解为一系列图片数据，并获取每帧的持续时间，然后通过动画图形动态加载帧图片。该方式适合播放从服务器获取的 GIF 文件。

从 GIF 文件中分解帧图片有现成的开源框架代码，具体参见本书的下载资源。下面是播放 GIF 动图的代码片段：

```
private void showGifAnimation() {
    ImageView iv_gif = (ImageView) findViewById(R.id.iv_gif);
    InputStream is = getResources().openRawResource(R.raw.welcome);
    GifImage gifImage = new GifImage();
    int code = gifImage.read(is);
    if (code == GifImage.STATUS_OK) {
        GifImage.GifFrame[] frameList = gifImage.getFrames();
        AnimationDrawable ad_gif = new AnimationDrawable();
        for (int i=0; i<frameList.length; i++) {
            //BitmapDrawable 用于把 Bitmap 格式转换为 Drawable 格式
            BitmapDrawable bd = new BitmapDrawable(getResources(), frameList[i].image);
            ad_gif.addFrame(bd, frameList[i].delay);
        }
        ad_gif.setOneShot(false);
        iv_gif.setImageDrawable(ad_gif);
        ad_gif.start();
    } else if (code == GifImage.STATUS_FORMAT_ERROR) {
        Toast.makeText(this, "该图片不是 gif 格式", Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(this, "gif 图片读取失败:" + code, Toast.LENGTH_LONG).show();
    }
}
```

GIF 文件的播放效果如图 12-4 和图 12-5 所示。其中，图 12-4 所示为 GIF 动图播放开始时的画面，图 12-5 所示为 GIF 动图临近播放结束时的画面。





图 12-4　GIF 动画开始播放



图 12-5　GIF 动画播放结束

12.1.3　淡入淡出动画

帧动画的帧显示方式采用后面一帧直接覆盖前面一帧，这在快速轮播时没什么问题，但是如果每帧的间隔时间比较长（比如超过 0.5 秒），两帧之间的画面切换就会很生硬，直接从前一帧变成后一帧会让人觉得很突兀。为了解决这种长间隔切换图片在视觉效果方面的问题，Android 提供了过渡图形 TransitionDrawable 处理两张图片之间的渐变显示，即淡入淡出的动画效果。

过渡图形同样需要宿主视图显示该图形，即调用 ImageView 对象的 setImageDrawable 方法进行图形加载操作。下面是 TransitionDrawable 的常用方法说明。

- 构造函数：指定过渡图形的图形数组。该图形数组大小为 2，包含前后两张图形。
- startTransition：开始过渡操作。这里需要先设置宿主视图，然后才能进行渐变显示。
- resetTransition：重置过渡操作。
- reverseTransition：倒过来执行过渡操作。

下面是使用过渡图形的代码片段：

```
private void showFadeAnimation() {
    //淡入淡出动画需要先设置一个 Drawable 数组，用于变换图片
    Drawable[] drawableArray = {
            getResources().getDrawable(R.drawable.fade_begin),
            getResources().getDrawable(R.drawable.fade_end)
            };
    TransitionDrawable td_fade = new TransitionDrawable(drawableArray);
    iv_fade_anim.setImageDrawable(td_fade);
    td_fade.startTransition(3000);
}
```

过渡图形的播放效果如图 12-6 和图 12-7 所示。其中，图 12-6 所示为开始转换不久的画面，此时仍以第一张图片为主；图 12-7 所示为转换将要结束的画面，此时已经基本过渡到第二张图片。



图 12-6　淡入淡出动画开始播放

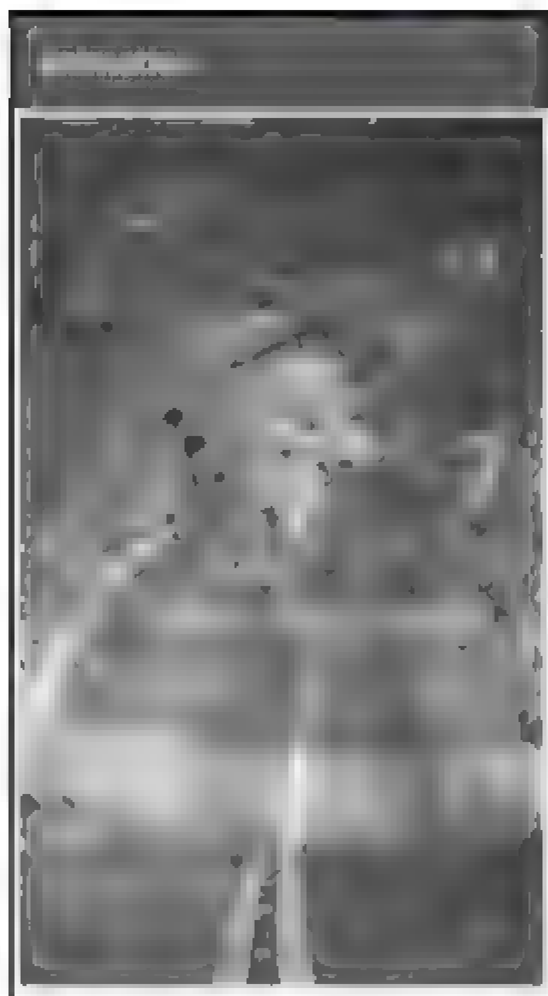


图 12-7　淡入淡出动画即将结束

至此，Android 的主要图形类都在本书做了介绍，为方便读者查阅，这里总结整理一下，简要说明见表 12-1。

表12-1　Android的主要图形类说明

Drawable 图形类	说明	XML 节点名称	参考章节
ColorDrawable	颜色图形	color	第 2 章的“2.1.2 颜色”
StateListDrawable	状态列表图形	selector	第 2 章的“2.4.2 状态列表图形”
ShapeDrawable	形状图形	shape	第 2 章的“2.4.3 形状图形”
GradientDrawable	渐变图形	gradient	第 2 章的“2.4.3 形状图形”
NinePatchDrawable	点九图形	nine-patch	第 2 章的“2.4.4 九宫格图片”
LayerDrawable	层次图形	layer-list	第 6 章的“6.4.2 进度条 ProcessBar”
ClipDrawable	裁剪图形	clip	第 6 章的“6.4.2 进度条 ProcessBar”
BitmapDrawable	位图图形	bitmap	第 11 章的“11.5.2 小知识：图像的基本加工”
AnimationDrawable	动画图形	animation-list	第 12 章的“12.1.1 帧动画的实现”
TransitionDrawable	过渡图形	transition	第 12 章的“12.1.3 淡入淡出动画”

12.2　补间动画

本节介绍补间动画的原理与用法，首先指出补间动画有四大类，分别是灰度动画、平移动画、缩放动画和旋转动画，介绍这 4 种动画的基本用法；接着阐述补间动画的原理，基于旋转动画的思想实现摇摆动画；然后介绍如何使用集合动画同时展示多种动画效果；最后就第 11 章的飞掠横幅遗留问题给出使用动画技术平滑切换前后视图的方案。

12.2.1 补间动画的种类

12.1 节提到两张图片之间的渐变效果可以使用过渡图形 TransitionDrawable 实现。一张图形内部能否运用渐变效果？比如对图片的大小进行自动缩放等。正好，Android 提供了补间动画，允许开发者实现某个视图的动态变换，具体包括 4 类动画效果，分别是灰度动画、平移动画、缩放动画和旋转动画。为什么把这 4 种动画称为补间动画呢？因为由开发者提供动画的起始状态值与终止状态值，然后系统按照时间推移计算中间的状态值，并自动把中间状态的视图补充到起止视图中，自动补充中间视图的动画就被简称为“补间动画”。

补间动画的 4 类动画（灰度动画 AlphaAnimation、平移动画 TranslateAnimation、缩放动画 ScaleAnimation 和旋转动画 RotateAnimation）都来自于共同的动画类 Animation，因此同时拥有 Animation 的属性与方法。下面是 Animation 的常用方法说明。

- setFillAfter：设置是否维持结束画面。true 表示动画结束后停留在结束画面，false 表示动画结束后恢复到开始画面。
- setRepeatMode：设置重播模式。Animation.RESTART 表示从头开始，Animation.REVERSE 表示倒过来开始。默认为 Animation.RESTART。
- setRepeatCount：设置重播次数。默认为 0 表示只播放一次。
- setDuration：设置动画的持续时间。单位毫秒。
- setInterpolator：设置动画的插值器。
- setAnimationListener：设置动画事件的监听器。需实现接口 AnimationListener 的 3 个方法。
    - ➢ onAnimationStart：在动画开始时触发。
    - ➢ onAnimationEnd：在动画结束时触发。
    - ➢ onAnimationRepeat：在动画重播时触发。

与帧动画一样，补间动画也需要找一个宿主视图，对宿主视图施展动画效果。不同的是，帧动画的宿主视图只能是 ImageView 相关的图像视图，而补间动画的宿主视图可以是任意视图，只要派生自 View 类就行。给补间动画指定宿主视图的方式很简单，调用宿主对象的 startAnimation 方法即可命令宿主视图开始动画，调用宿主对象的 clearAnimation 方法即可要求宿主视图清除动画。

具体到每种补间动画又有不同的初始化方式。下面来看具体说明。

（1）初始化灰度动画：在构造函数中指定视图透明度的前后数值。取值为 0.0～1.0，0 表示完全不透明，1 表示完全透明。

（2）初始化平移动画：在构造函数中指定视图左上角在平移前后的坐标值。其中，第一个参数为平移前的横坐标，第二个参数为平移后的横坐标，第三个参数为平移前的纵坐标，第四个参数为平移后的纵坐标。

（3）初始化缩放动画：在构造函数中指定视图横纵坐标的前后缩放比例。缩放比例取值 0.5 表示缩小到原来的二分之一，取值 2 表示放大到原来的两倍。其中，第一个参数为缩放前的横坐标比例，第二个参数为缩放后的横坐标比例，第三个参数为缩放前的纵坐标比例，第四个参数为缩放后的纵坐标比例。



（4）初始化旋转动画：在构造函数中指定视图的旋转角度。其中，第一个参数为旋转前的角度，第二个参数为旋转后的角度，第三个参数为圆心的横坐标类型，第四个参数为圆心横坐标的数值比例，第五个参数为圆心的纵坐标类型，第六个参数为圆心纵坐标的数值比例。坐标类型的取值说明见表 12-2。

表12-2　坐标类型的取值说明

Animation 类的坐标类型	说明
ABSOLUTE	绝对位置
RELATIVE_TO_SELF	相对自身位置
RELATIVE_TO_PARENT	相对上级位置

下面是使用 4 种补间动画的代码：

```
public class TweenAnimActivity extends AppCompatActivity implements AnimationListener {
    private ImageView iv_tween_anim;
    private Animation alphaAnim, translateAnim, scaleAnim, rotateAnim;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tween_anim);
        iv_tween_anim = (ImageView) findViewById(R.id.iv_tween_anim);
        initAnim();
        initTweenSpinner();
    }

    private void initAnim() {
        // 从完全透明变为即将不透明
        alphaAnim = new AlphaAnimation(1.0f, 0.1f);
        alphaAnim.setDuration(3000);
        alphaAnim.setFillAfter(true);
        // 向左平移 200
        translateAnim = new TranslateAnimation(1.0f, -200f, 1.0f, 1.0f);
        translateAnim.setDuration(3000);
        translateAnim.setFillAfter(true);
        // 宽度不变，高度变为原来的二分之一
        scaleAnim = new ScaleAnimation(1.0f, 1.0f, 1.0f, 0.5f);
        scaleAnim.setDuration(3000);
        scaleAnim.setFillAfter(true);
        // 围绕着圆心顺时针旋转 360 度
        rotateAnim = new RotateAnimation(0f, 360f, Animation.RELATIVE_TO_SELF,
                0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
        rotateAnim.setDuration(3000);
```



```
        rotateAnim.setFillAfter(true);
    }

    private void initTweenSpinner() {
        ArrayAdapter<String> tweenAdapter = new ArrayAdapter<String>(this,
                R.layout.item_select, tweenArray);
        Spinner sp_tween = (Spinner) findViewById(R.id.sp_tween);
        sp_tween.setPrompt("请选择补间动画类型");
        sp_tween.setAdapter(tweenAdapter);
        sp_tween.setOnItemSelectedListener(new TweenSelectedListener());
        sp_tween.setSelection(0);
    }

    private String[] tweenArray={"灰度动画", "平移动画", "缩放动画", "旋转动画"};
    class TweenSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            if (arg2 == 0) {
                iv_tween_anim.startAnimation(alphaAnim);
                alphaAnim.setAnimationListener(TweenAnimActivity.this);
            } else if (arg2 == 1) {
                iv_tween_anim.startAnimation(translateAnim);
                translateAnim.setAnimationListener(TweenAnimActivity.this);
            } else if (arg2 == 2) {
                iv_tween_anim.startAnimation(scaleAnim);
                scaleAnim.setAnimationListener(TweenAnimActivity.this);
            } else if (arg2 == 3) {
                iv_tween_anim.startAnimation(rotateAnim);
                rotateAnim.setAnimationListener(TweenAnimActivity.this);
            }
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }

    @Override
    public void onAnimationStart(Animation animation) {
    }

    @Override
    public void onAnimationEnd(Animation animation) {
        if (animation.equals(alphaAnim)) {
            Animation alphaAnim2 = new AlphaAnimation(0.1f, 1.0f);
```

```
                alphaAnim2.setDuration(3000);
                alphaAnim2.setFillAfter(true);
                iv_tween_anim.startAnimation(alphaAnim2);
            } else if (animation.equals(translateAnim)) {
                Animation translateAnim2 = new TranslateAnimation(-200f, 1.0f, 1.0f, 1.0f);
                translateAnim2.setDuration(3000);
                translateAnim2.setFillAfter(true);
                iv_tween_anim.startAnimation(translateAnim2);
            } else if (animation.equals(scaleAnim)) {
                Animation scaleAnim2 = new ScaleAnimation(1.0f, 1.0f, 0.5f, 1.0f);
                scaleAnim2.setDuration(3000);
                scaleAnim2.setFillAfter(true);
                iv_tween_anim.startAnimation(scaleAnim2);
            } else if (animation.equals(rotateAnim)) {
                Animation rotateAnim2 = new RotateAnimation(0f, -360f,
                        Animation.RELATIVE_TO_SELF, 0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
                rotateAnim2.setDuration(3000);
                rotateAnim2.setFillAfter(true);
                iv_tween_anim.startAnimation(rotateAnim2);
            }
        }

        @Override
        public void onAnimationRepeat(Animation animation) {
        }
    }
```

补间动画的播放效果如图 12-8～图 12-15 所示。其中，图 12-8 和图 12-9 所示为灰度动画播放前后的画面，图 12-10 和图 12-11 所示为平移动画播放前后的画面，图 12-12 和图 12-13 所示为缩放动画播放前后的画面，图 12-14 和图 12-15 所示为旋转动画播放前后的画面。



图 12-8　灰度动画开始播放



图 12-9　灰度动画即将结束



图 12-10 平移动画开始播放

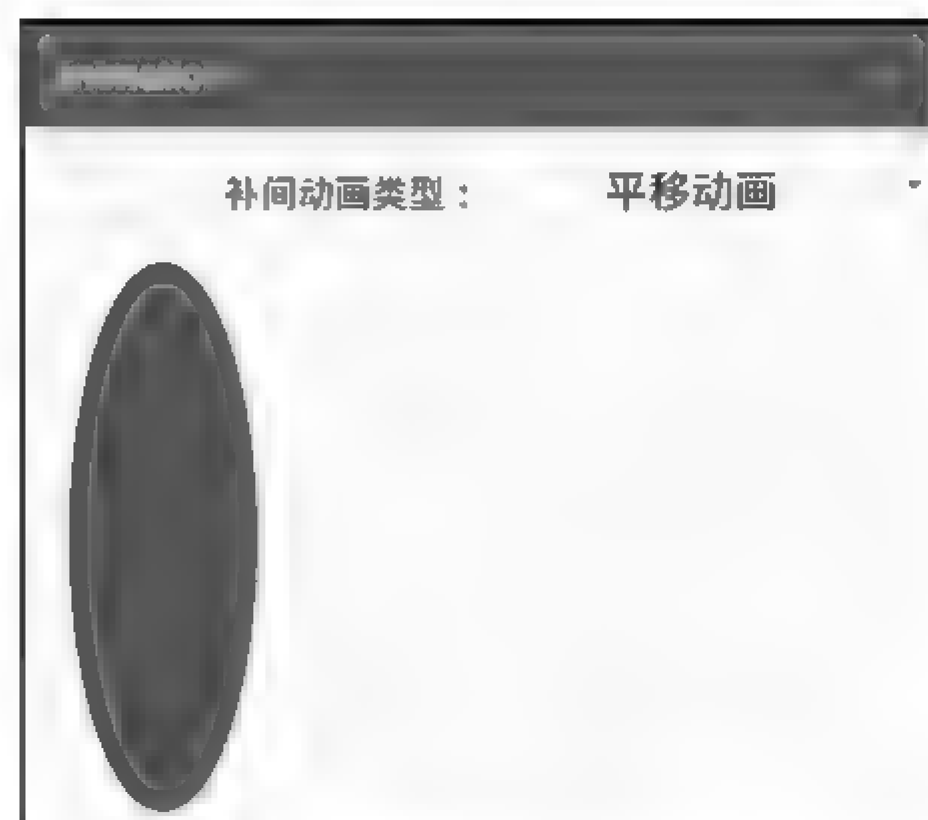


图 12-11 平移动画即将结束

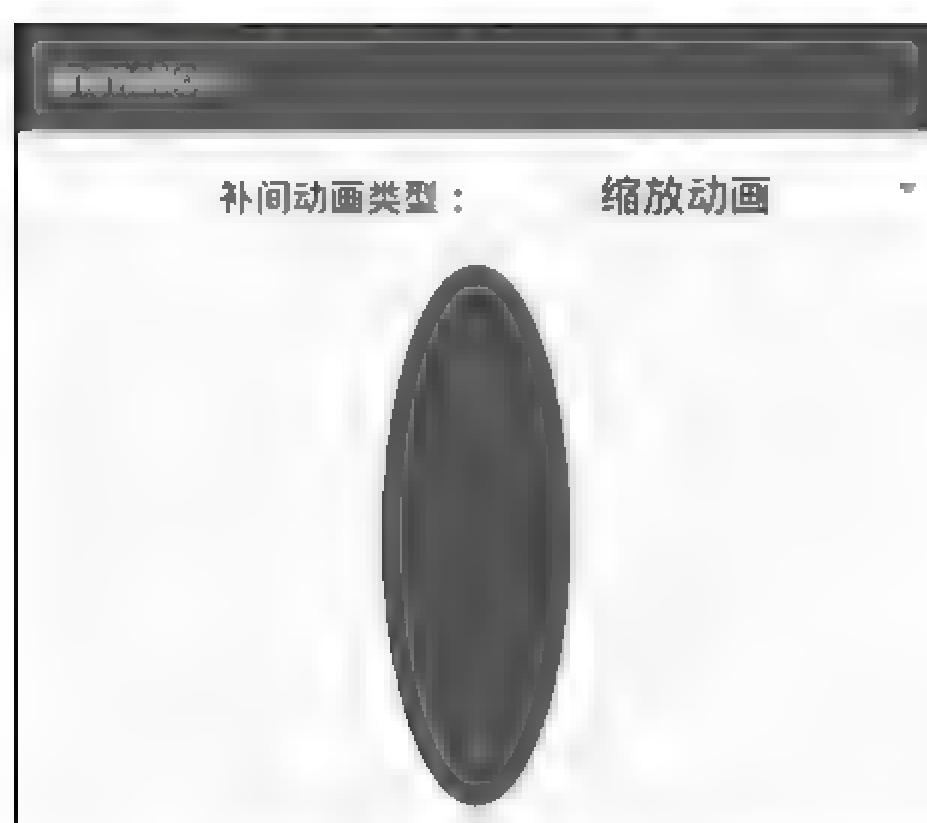


图 12-12 缩放动画开始播放



图 12-13 缩放动画即将结束

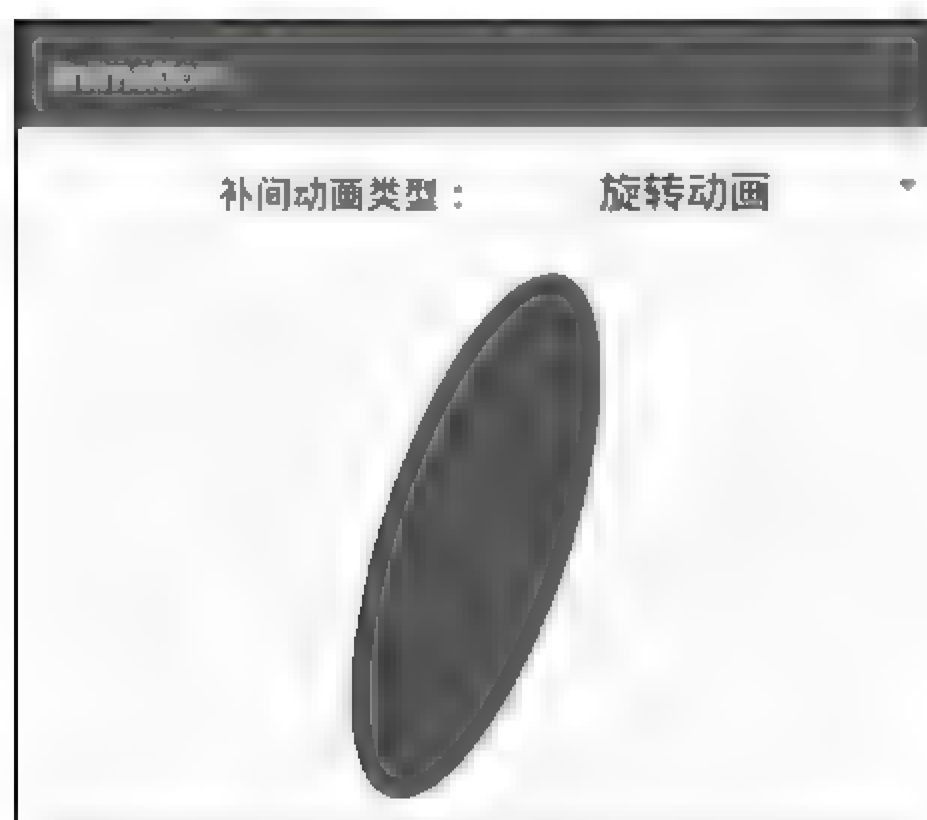


图 12-14 旋转动画开始播放

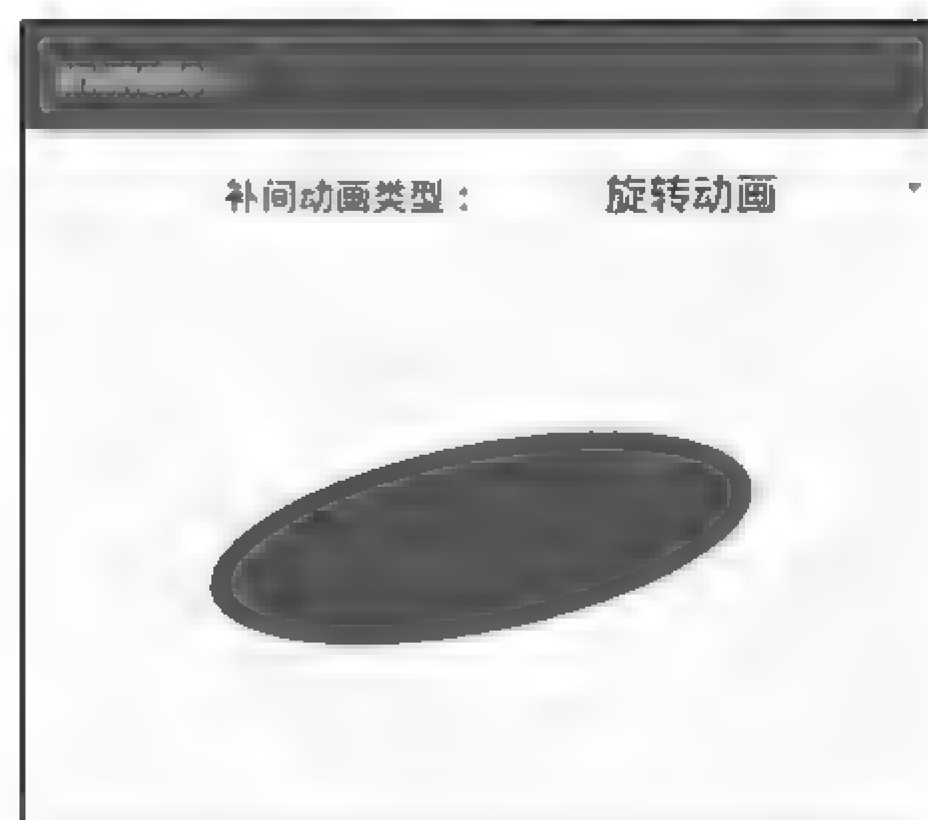


图 12-15 旋转动画正在播放

12.2.2 补间动画的原理

补间动画只提供了基本的动态变换，如果想要复杂的动画效果，比如像钟摆一样左摆一



下再右摆一下，补间动画就无能为力了。我们有必要了解补间动画的实现原理，这样才能进行适当的改造，使其符合实际的业务需求。

下面以旋转动画 RotateAnimation 为例说明补间动画的实现原理。查看 RotateAnimation 的源码，发现除了一堆构造函数外，剩下的代码只有3个函数：

```
private void initializePivotPoint() {
    if (mPivotXType == ABSOLUTE) {
        mPivotX = mPivotXValue;
    }
    if (mPivotYType == ABSOLUTE) {
        mPivotY = mPivotYValue;
    }
}

@Override
protected void applyTransformation(float interpolatedTime, Transformation t) {
    float degrees = mFromDegrees + ((mToDegrees - mFromDegrees) * interpolatedTime);
    float scale = getScaleFactor();

    if (mPivotX == 0.0f && mPivotY == 0.0f) {
        t.getMatrix().setRotate(degrees);
    } else {
        t.getMatrix().setRotate(degrees, mPivotX * scale, mPivotY * scale);
    }
}

@Override
public void initialize(int width, int height, int parentWidth, int parentHeight) {
    super.initialize(width, height, parentWidth, parentHeight);
    mPivotX = resolveSize(mPivotXType, mPivotXValue, width, parentWidth);
    mPivotY = resolveSize(mPivotYType, mPivotYValue, height, parentHeight);
}
```

注意两个初始化函数都在处理圆心的坐标，实际与动画播放有关的代码只有 applyTransformation 方法。该方法很简单，提供了两个输入参数，第一个参数为插值时间，即逝去的时间所占的百分比，第二个参数为转换器。方法内部根据插值时间计算当前所处的角度 degrees，最后使用转换器把视图旋转到该角度。

查看其他补间动画的源码，发现都与 RotateAnimation 的处理大同小异，对中间状态的视图变换处理不外乎以下两个步骤：

步骤01 根据插值时间计算当前的状态值（如灰度、距离、比率、角度等）。

步骤02 在宿主视图上使用该状态值进行变换操作。



如此看来，补间动画的关键在于利用插值时间计算状态值。现在回头看看钟摆的左右摆动，这个摆动操作其实由 3 段旋转动画构成。

（1）以上面的端点为圆心，钟摆以垂直向下的状态向左旋转，转到左边的某个角度停住（比如左转 60 度）。

（2）钟摆从左边向右边旋转，转到右边的某个角度停住（比如右转 120 度，与垂直方向的夹角为 60 度）。

（3）钟摆从右边再向左旋转，当其摆到垂直方向时，完成一个周期的摇摆动作。

弄清楚了摇摆动画的运动过程，接下来根据插值时间计算对应的角度。具体到代码实现上，需要做以下两处调整：

（1）旋转动画初始化时只有两个度数，即起始角度和终止角度。摇摆动画需要 3 个参数，即中间角度（既是起始角度也是终止角度）、摆到左侧的角度和摆到右侧的角度。

（2）根据插值时间估算当前所处的角度。对于摇摆动画来说，需要做 3 个分支判断（对应之前 3 段旋转动画）。如果整个动画持续 4 秒，那么 0～1 秒为往左的旋转动画，该区间的起始角度为中间角度，终止角度为摆到左侧的角度；1～3 秒为往右的旋转动画，该区间的起始角度为摆到左侧的角度，终止角度为摆到右侧的角度；3～4 秒为往左的旋转动画，该区间的起始角度为摆到右侧的角度，终止角度为中间角度。

分析完毕，贴上修改后的摇摆动画代码片段：

```
protected void applyTransformation(float interpolatedTime, Transformation t) {
        float degrees;
        float leftPos = (float) (1.0 / 4.0);
        float rightPos = (float) (3.0 / 4.0);
        if (interpolatedTime <= leftPos) {
                degrees = mMiddleDegrees + ((mLeftDegrees - mMiddleDegrees) * interpolatedTime * 4);
        } else if (interpolatedTime > leftPos && interpolatedTime < rightPos) {
                degrees = mLeftDegrees + ((mRightDegrees-mLeftDegrees) * (interpolatedTime-leftPos) * 2);
        } else {
                degrees = mRightDegrees+ ((mMiddleDegrees-mRightDegrees)*(interpolatedTime-rightPos)*4);
        }

        float scale = getScaleFactor();
        if (mPivotX == 0.0f && mPivotY == 0.0f) {
                t.getMatrix().setRotate(degrees);
        } else {
                t.getMatrix().setRotate(degrees, mPivotX * scale, mPivotY * scale);
        }
}
```

摇摆动画的播放效果如图 12-16 和图 12-17 所示。其中，图 12-16 所示为钟摆向左摆动时的画面，图 12-17 所示为钟摆向右摆动时的画面。



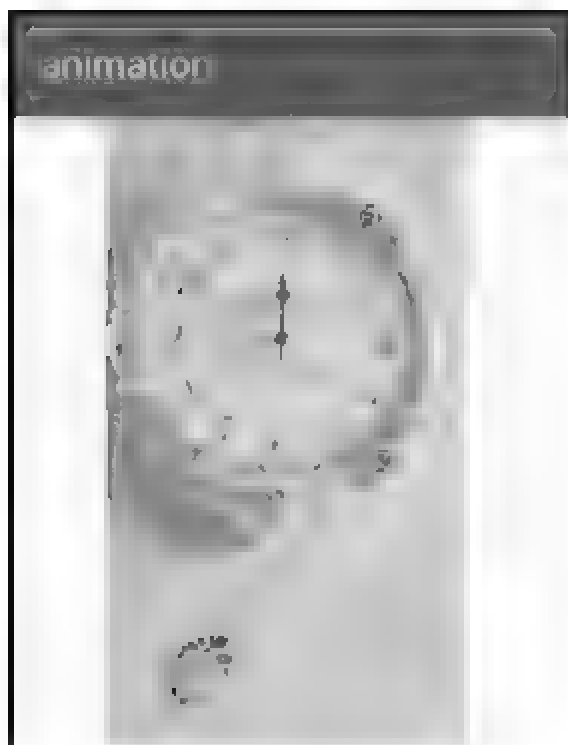


图 12-16　摇摆动画向左摆动

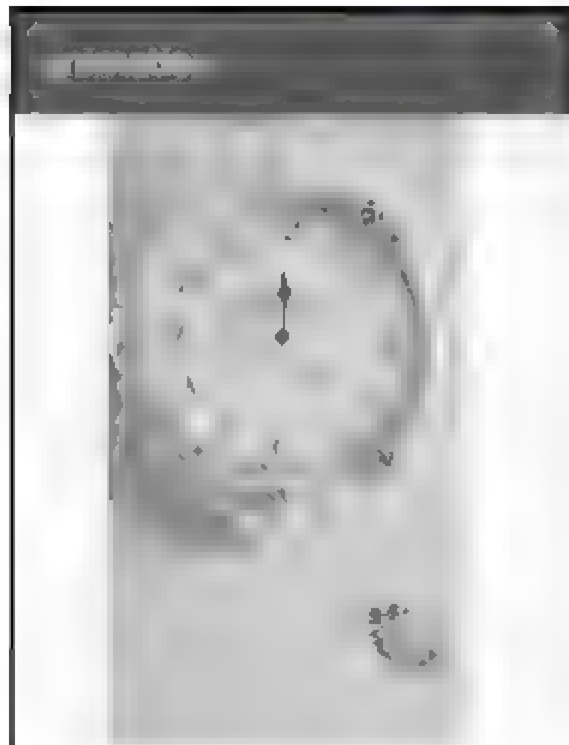


图 12-17　摇摆动画向右摆动

12.2.3　集合动画

有时，一个动画效果会揉合多种动画技术，比如一边旋转、一边缩放用到集合动画 AnimationSet，把几个补间动画组装起来，实现让某视图同时呈现多种动画的效果。

集合动画与补间动画一样继承自 Animation 类，所以拥有补间动画的基本方法。但集合动画不像一般补间动画那样提供构造函数，而是通过 addAnimation 方法把别的补间动画加入本集合动画中。

下面是使用集合动画的代码片段：

```
private void initAnim() {
    alphaAnim = new AlphaAnimation(1.0f, 0.1f);   // 灰度动画
    alphaAnim.setDuration(3000);
    alphaAnim.setFillAfter(true);
    translateAnim = new TranslateAnimation(1.0f, -200f, 1.0f, 1.0f);   // 平移动画
    translateAnim.setDuration(3000);
    translateAnim.setFillAfter(true);
    scaleAnim = new ScaleAnimation(1.0f, 1.0f, 1.0f, 0.5f);   // 缩放动画
    scaleAnim.setDuration(3000);
    scaleAnim.setFillAfter(true);
    rotateAnim = new RotateAnimation(0f,   360f, Animation.RELATIVE_TO_SELF,
            0.5f, Animation.RELATIVE_TO_SELF, 0.5f);   // 旋转动画
    rotateAnim.setDuration(3000);
    rotateAnim.setFillAfter(true);
    setAnim = new AnimationSet(true);   // 集合动画
    setAnim.addAnimation(translateAnim);
    setAnim.addAnimation(alphaAnim);
    setAnim.addAnimation(scaleAnim);
    setAnim.addAnimation(rotateAnim);
    setAnim.setFillAfter(true);
    iv_anim_set.startAnimation(setAnim);
```

```
            setAnim.setAnimationListener(this);
        }
```

集合动画的播放效果如图 12-18 和图 12-19 所示。其中，图 12-18 所示为集合动画开始不久的画面，图 12-19 所示为集合动画即将结束的画面。

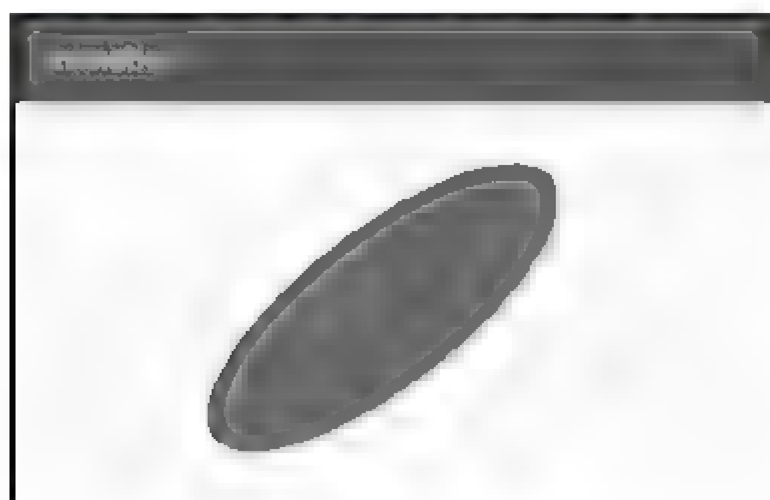


图 12-18　集合动画开始播放不久

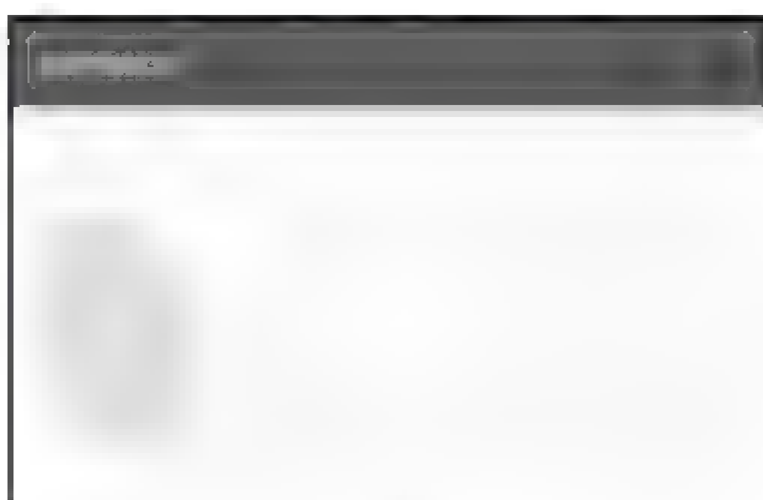


图 12-19　集合动画即将结束播放

帧动画允许在 XML 文件中存放动画定义，补间动画也允许，就连集合动画都可以放在一块描述。下面是一个集合动画的 XML 文件定义的例子，其中包含 4 个补间动画定义：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha android:duration="3000" android:fromAlpha="1.0" android:toAlpha="0.1" />
    <translate android:duration="3000" android:fromXDelta="1.0" android:toXDelta="-200"
        android:fromYDelta="1.0" android:toYDelta="1.0" />
    <scale android:duration="3000"   android:fromXScale="1.0" android:toXScale="1.0"
        android:fromYScale="1.0" android:toYScale="0.5" />
    <rotate android:duration="3000" android:fromDegrees="0" android:toDegrees="360"
        android:pivotX="50%" android:pivotY="50%" />
</set>
```

在代码中调用动画工具 AnimationUtils 的 loadAnimation 方法，即可加载该集合动画的文件定义，无须在代码中定义其他 4 种补间动画。具体加载代码如下：

```
            setAnim.addAnimation(AnimationUtils.loadAnimation(this, R.anim.anim_set));
```

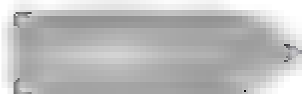
使用上述 XML 文件演示集合动画的效果如图 12-18 和图 12-19 所示，画面效果与代码定义方式没什么区别。

12.2.4　在飞掠横幅中使用补间动画

第 11 章介绍飞掠视图 ViewFlipper 时，结合手势检测器 GestureDetector 实现了飞掠横幅的效果。不过前后 Banner 的飞掠切换有些生硬，后面的广告图一下子把前面的广告图覆盖，显得十分突兀，完全不如 ViewPager 那样翻页自然。现在我们正好活学活用，试试利用补间动画技术给飞掠横幅加上动画翻页变换，看看能否达到自然翻页的预期效果。

第 11 章提到，ViewFlipper 有以下 4 个操作动画的方法：

- setInAnimation：设置视图的移入动画。
- getInAnimation：获取移入动画的动画对象。



- setOutAnimation：设置视图的移出动画。
- getOutAnimation：获取移出动画的动画对象。

通过这 4 个动画方法加载动画定义，应该能实现飞掠视图前后切换的动画效果。

首先定义几个动画定义文件，用来描述移入动画和移出动画的行为。具体地说，包括 4 个动画定义文件：向左移入动画、向左移出动画、向右移入动画和向右移出动画。下面对这 4 个动画定义文件分别进行说明。

（1）向左移入动画，用来描述 Banner 向左翻页时右边页面的移入行为，动画文件名为 push_left_in.xml，文件内容如下：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:duration="1500" android:fromXDelta="100.0%p" android:toXDelta="0.0" />
    <alpha android:duration="1500" android:fromAlpha="0.1" android:toAlpha="1.0" />
</set>
```

（2）向左移出动画，用来描述 Banner 向左翻页时左边页面的移出行为，动画文件名为 push_left_out.xml，文件内容如下：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:duration="1500" android:fromXDelta="0.0" android:toXDelta="-100.0%p" />
    <alpha android:duration="1500" android:fromAlpha="1.0" android:toAlpha="0.1" />
</set>
```

（3）向右移入动画，用来描述 Banner 向右翻页时左边页面的移入行为，动画文件名为 push_right_in.xml，文件内容如下：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:duration="1500" android:fromXDelta="-100.0%p" android:toXDelta="0.0" />
    <alpha android:duration="1500" android:fromAlpha="0.1" android:toAlpha="1.0" />
</set>
```

（4）向右移出动画，用来描述 Banner 向右翻页时右边页面的移出行为，动画文件名为 push_right_out.xml，文件内容如下：

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate android:duration="1500" android:fromXDelta="0.0" android:toXDelta="100.0%p" />
    <alpha android:duration="1500" android:fromAlpha="1.0" android:toAlpha="0.1" />
</set>
```

在第 11 章的 BannerFlipper 代码中补充以下片段，加载相关动画定义文件，并在翻页时展示动画：

```
    private void startFlip() {
        mFlipper.startFlipping();
        mFlipper.setInAnimation(AnimationUtils.loadAnimation(mContext, R.anim.push_left_in));
        mFlipper.setOutAnimation(AnimationUtils.loadAnimation(mContext, R.anim.push_left_out));
        mFlipper.getOutAnimation().setAnimationListener(new BannerAnimationListener(this));
```



```
            mFlipper.showNext();
        }

        private void backFlip() {
            mFlipper.startFlipping();
            mFlipper.setInAnimation(AnimationUtils.loadAnimation(mContext, R.anim.push_right_in));
            mFlipper.setOutAnimation(AnimationUtils.loadAnimation(mContext, R.anim.push_right_out));
            mFlipper.getOutAnimation().setAnimationListener(new BannerAnimationListener(this));
            mFlipper.showPrevious();
            mFlipper.setInAnimation(AnimationUtils.loadAnimation(mContext, R.anim.push_left_in));
            mFlipper.setOutAnimation(AnimationUtils.loadAnimation(mContext, R.anim.push_left_out));
            mFlipper.getOutAnimation().setAnimationListener(new BannerAnimationListener(this));
        }

        private class BannerAnimationListener implements Animation.AnimationListener {
            private BannerAnimationListener(BannerFlipper bannerFlipper) {
            }

            @Override
            public final void onAnimationEnd(Animation paramAnimation) {
                int position = mFlipper.getDisplayedChild();
                ((RadioButton) mGroup.getChildAt(position)).setChecked(true);
            }

            @Override
            public final void onAnimationRepeat(Animation paramAnimation) {
            }

            @Override
            public final void onAnimationStart(Animation paramAnimation) {
            }
        }
```

改造后的飞掠横幅在翻页时的动画效果如图 12-20 和图 12-21 所示。其中，图 12-20 所示为向左翻页开始不久的画面，右边页面逐步移入且色彩渐渐淡入；图 12-21 所示为向左翻页即将结束时的画面，左边页面逐步移出且色彩渐渐淡出。

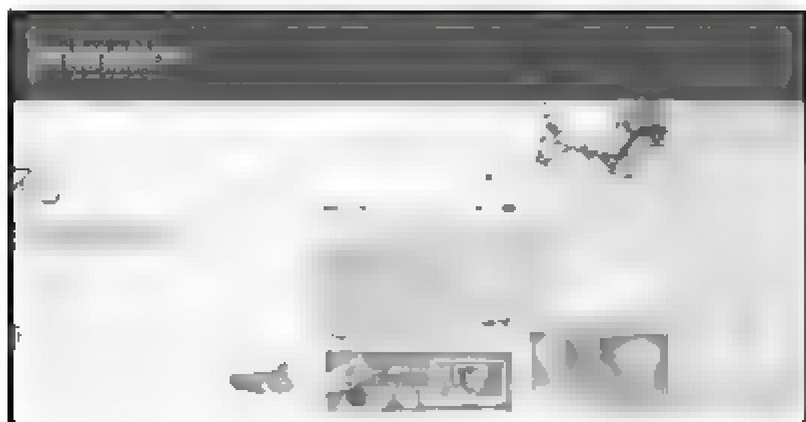


图 12-20　飞掠横幅开始向左翻页

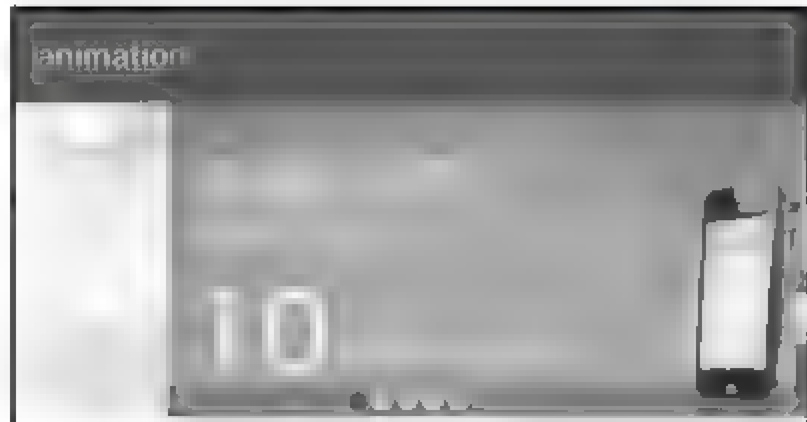


图 12-21　飞掠横幅左翻即将结束

读者是否注意到，集成了动画效果的飞掠横幅与第 7 章的横幅轮播 Banner 竟有几分相似。采用不同技术实现的效果殊途同归，这正是 Android 开发的魅力所在。



12.3　属性动画

本节介绍属性动画的应用场合与进阶用法，首先说明为何属性动画是补间动画的升级版，以及属性动画的基本用法；接着说明如何运用属性动画组合实现多个属性动画的同时播放与顺序播放效果；最后对动画技术中的插值器和估值器进行分析，并演示不同插值器的动画效果。

12.3.1　属性动画的用法

视图 View 有许多状态属性，4 种补间动画只对其中 6 种属性进行操作，这 6 种属性的说明见表 12-3。

表12-3　补间动画的属性说明

View 类的属性名称	属性说明	属性设置方法	对应的补间动画
alpha	透明度	setAlpha	灰度动画
rotation	旋转角度	setRotation	旋转动画
scaleX	横坐标的缩放比例	setScaleX	缩放动画
scaleY	纵坐标的缩放比例	setScaleY	缩放动画
translationX	横坐标的平移距离	setTranslationX	平移动画
translationY	纵坐标的平移距离	setTranslationY	平移动画

每个控件的属性远不止这 6 种，如果要求对视图的背景颜色做渐变处理，补间动画就无能为力了。为此，Android 自 3.0 后引入了属性动画 ObjectAnimator，属性动画突破了补间动画的局限，允许视图的所有属性都能实现渐变的动画效果，例如背景颜色、文字颜色、文字大小等。只要设定某属性的起始值与终止值、渐变的持续时间，属性动画即可实现该属性的动画渐变效果。

下面是 ObjectAnimator 的常用方法。

- ofInt：定义整型属性的属性动画。
- ofFloat：定义浮点型属性的属性动画。
- ofArgb：定义颜色属性的属性动画。
- ofObject：定义对象属性的属性动画。用于不是上述三种类型的属性，例如 Rect 对象。

以上 4 个 of 方法的第一个参数为宿主视图对象，第二个参数为需要变化的属性名称，第三个参数后为属性变化的各个状态值。注意，of 方法后面的参数个数是变化的。如果第 3 个参数是状态 A，第 4 个参数是状态 B，属性动画就从 A 状态变为 B 状态；如果第 3 个参数是状态 A，第 4 个参数是状态 B，第 5 个参数是状态 C，属性动画就先从 A 状态变为 B 状态，再从 B 状态变为 C 状态。

- setRepeatMode：设置重播模式。ValueAnimator.RESTART 表示从头开始，ValueAnimator.REVERSE 表示倒过来开始。默认为 ValueAnimator.RESTART。



- setRepeatCount：设置重播次数。默认为 0 表示只播放一次。
- setDuration：设置动画的持续时间。单位毫秒。
- setInterpolator：设置动画的插值器。
- setEvaluator：设置动画的估值器。
- start：开始播放动画。
- cancel：取消播放动画。
- end：结束播放动画。
- pause：暂停播放动画。
- resume：恢复播放动画。
- reverse：倒过来播放动画。
- isRunning：判断动画是否在播放。注意，暂停时，isRunning 方法仍然返回 true。
- isPaused：判断动画是否被暂停。
- isStarted：判断动画是否已经开始。注意，曾经播放与正在播放都算已经开始。
- addListener：添加动画监听器，需实现接口 AnimatorListener 的 4 个方法。
    - onAnimationStart：在动画开始播放时触发。
    - onAnimationEnd：在动画结束播放时触发。
    - onAnimationCancel：在动画取消播放时触发。
    - onAnimationRepeat：在动画重播时触发。
- removeListener：移出指定的动画监听器。
- removeAllListeners：移出所有动画监听器。

下面是使用属性动画的代码：

```
public class ObjectAnimActivity extends AppCompatActivity {
    private ImageView iv_object_anim;
    private ObjectAnimator alphaAnim, translateAnim, scaleAnim, rotateAnim, clipAnim;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_object_anim);
        iv_object_anim = (ImageView) findViewById(R.id.iv_object_anim);
        initAnim();
        initObjectSpinner();
    }

    private void initAnim() {
        alphaAnim = ObjectAnimator.ofFloat(iv_object_anim, "alpha", 1f, 0.1f, 1f);
        translateAnim = ObjectAnimator.ofFloat(iv_object_anim, "translationX", 0f, -200f, 0f, 200f, 0f);
        scaleAnim = ObjectAnimator.ofFloat(iv_object_anim, "scaleY", 1f, 0.5f, 1f);
        rotateAnim = ObjectAnimator.ofFloat(iv_object_anim, "rotation", 0f, 360f, 0f);
    }
```

```
private void initObjectSpinner() {
    ArrayAdapter<String> objectAdapter = new ArrayAdapter<String>(this,
            R.layout.item_select, objectArray);
    Spinner sp_object = (Spinner) findViewById(R.id.sp_object);
    sp_object.setPrompt("请选择属性动画类型");
    sp_object.setAdapter(objectAdapter);
    sp_object.setOnItemSelectedListener(new ObjectSelectedListener());
    sp_object.setSelection(0);
}

private String[] objectArray = {"灰度动画", "平移动画", "缩放动画", "旋转动画", "裁剪动画"};
class ObjectSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        showAnimation(arg2);
    }

    public void onNothingSelected(AdapterView<?> arg0) {
    }
}

@TargetApi(Build.VERSION_CODES.JELLY_BEAN_MR2)
private void showAnimation(int type) {
    ObjectAnimator anim = null;
    if (type == 0) {
        anim = alphaAnim;
    } else if (type == 1) {
        anim = translateAnim;
    } else if (type == 2) {
        anim = scaleAnim;
    } else if (type == 3) {
        anim = rotateAnim;
    } else if (type == 4) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2) {
            int width = iv_object_anim.getWidth();
            int height = iv_object_anim.getHeight();
            clipAnim = ObjectAnimator.ofObject(iv_object_anim, "clipBounds",
                    new RectEvaluator(), new Rect(0,0,width,height),
                    new Rect(width/3,height/3,width/3*2,height/3*2),
                    new Rect(0,0,width,height));
            anim = clipAnim;
        } else {
            Toast.makeText(this,
                        "裁剪动画要求 Android 为 4.3 以上版本", Toast.LENGTH_SHORT).show();
            return;
        }
    }
```

```
                if (anim != null) {
                    anim.setDuration(3000);
                    anim.start();
                }
            }
        }
```

在上述代码演示的属性动画中，补间动画已经实现的效果就不再给出图示了，补间动画未实现的裁剪动画效果如图 12-22 和图 12-23 所示。其中，图 12-22 所示为裁剪即将开始时的画面，图 12-23 所示为裁剪过程中的画面。



图 12-22　裁剪动画即将开始



图 12-23　裁剪动画正在播放

12.3.2　属性动画组合

补间动画可以通过集合动画 AnimationSet 组装多种动画效果，属性动画也有类似的做法，即通过属性动画组合 AnimatorSet 组装多种属性动画。

AnimatorSet 虽然与 ObjectAnimator 都是继承自 Animator，但是两者的使用方法略有出入，主要是属性动画组合少了部分方法。下面是 AnimatorSet 的常用方法。

- setDuration：设置动画组合的持续时间。单位毫秒。
- setInterpolator：设置动画组合的插值器。
- play：设置当前动画。该方法返回一个 AnimatorSet.Builder 对象，可对该对象调用组装方法添加新动画，从而实现动画组装功能。下面是 Builder 的组装方法说明。
    - with：指定该动画与当前动画一起播放。
    - before：指定该动画在当前动画之前播放。
    - after：指定该动画在当前动画之后播放。
- start：开始播放动画组合。
- pause：暂停播放动画组合。
- resume：恢复播放动画组合。
- cancel：取消播放动画组合。

- end：结束播放动画组合。
- isRunning：判断动画组合是否在播放。
- isStarted：判断动画组合是否已经开始。

下面是使用属性动画组合的代码：

```
private void initAnim() {
    ObjectAnimator anim1 = ObjectAnimator.ofFloat(iv_object_group, "translationX", 0f, 100f);
    ObjectAnimator anim2 = ObjectAnimator.ofFloat(iv_object_group, "alpha", 1f, 0.1f, 1f, 0.5f, 1f);
    ObjectAnimator anim3 = ObjectAnimator.ofFloat(iv_object_group, "rotation", 0f, 360f);
    ObjectAnimator anim4 = ObjectAnimator.ofFloat(iv_object_group, "scaleY", 1f, 0.5f, 1f);
    ObjectAnimator anim5 = ObjectAnimator.ofFloat(iv_object_group, "translationX", 100f, 0f);
    animSet = new AnimatorSet();
    AnimatorSet.Builder builder = animSet.play(anim2);
    // anim1 先执行，然后再同步执行 anim2、anim3、anim3，最后执行 anim5
    builder.with(anim3).with(anim4).after(anim1).before(anim5);
    animSet.setDuration(4500);
    animSet.start();
}
```

属性动画组合的演示效果如图 12-24 和图 12-25 所示。其中，图 12-24 所示为动画组合开始播放不久的画面，图 12-25 所示为动画组合播放过程中的画面。

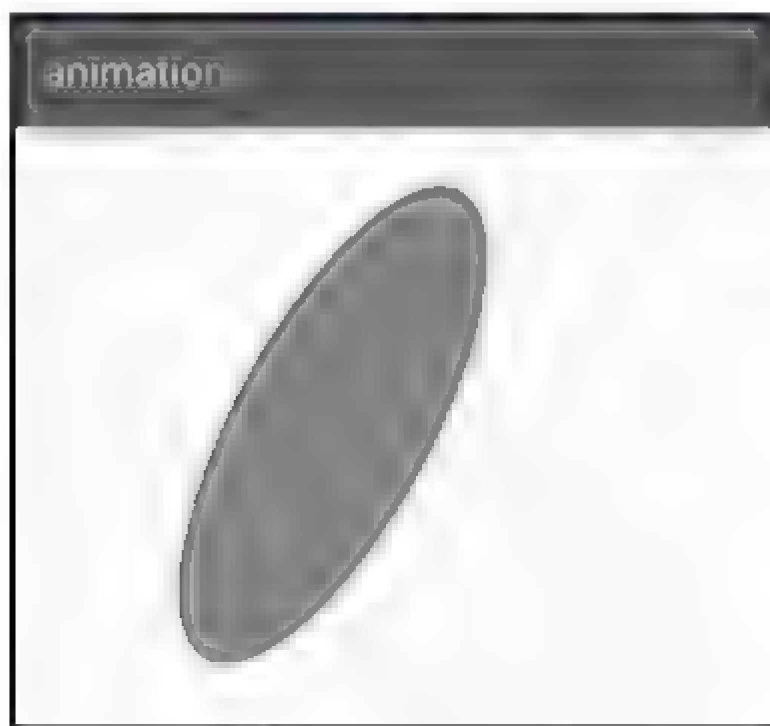


图 12-24　属性动画组合开始播放


图 12-25　属性动画组合正在播放

12.3.3　插值器和估值器

前面在介绍补间动画与属性动画时提到了插值器，属性动画还提到了估值器，因为插值器和估值器是相互关联的，所以放到一起介绍。

插值器用来控制属性值的变化速率，也可以理解为动画播放的速度，默认是匀速播放。要给动画播放指定某种速率形式，调用 setInterpolator 方法设置对应的插值器实现类即可，无论是补间动画、集合动画、属性动画，还是属性动画组合，都可以设置插值器。插值器实现类的说明见表 12-4。

表12-4　插值器实现类的说明

插值器的实现类	说明
LinearInterpolator	匀速插值器
AccelerateInterpolator	加速插值器
DecelerateInterpolator	减速插值器
AccelerateDecelerateInterpolator	落水插值器，即前半段加速、后半段减速
AnticipateInterpolator	射箭插值器，后退几步再往前冲
OvershootInterpolator	回旋插值器，冲过头再归位
AnticipateOvershootInterpolator	射箭回旋插值器，后退几步再往前冲，冲过头再归位
BounceInterpolator	震荡插值器，类似皮球落地（落地后会弹起几次）
CycleInterpolator	钟摆插值器，以开始位置为中线而晃动（类似摇摆动画，开始位置与结束位置的距离就是摇摆的幅度）

估值器专用于属性动画，主要描述该属性的数值变化要采用什么单位，比如整型数的渐变数值要取整，颜色的渐变数值为 ARGB 格式的颜色对象，矩形的渐变数值为 Rect 对象等。要给属性动画设置估值器，调用属性动画对象的的 setEvaluator 方法即可。估值器实现类的说明见表 12-5。

表12-5　估值器实现类的说明

估值器的实现类	说明
IntEvaluator	整型估值器
FloatEvaluator	浮点型估值器
ArgbEvaluator	颜色估值器
RectEvaluator	矩形估值器

一般情况下，无须单独设置属性动画的估值器，使用系统默认的估值器即可。但是如果属性类型不是 int、float、argb 三种，只能通过 ofObject 方法构造属性动画对象，就必须指定该属性的估值器，否则系统不知道如何计算渐变属性值。为方便记忆属性动画的构造方法与估值器的关联关系，表 12-6 列出了两者之间的对应关系。

表12-6　属性类型与估值器的对应关系

属性动画的构造方法	估值器	对应的属性说明
ofInt	IntEvaluator	整型类型的属性
ofFloat	FloatEvaluator	大部分状态属性，如 alpha、rotation、scaleY、translationX、textSize 等
ofArgb	ArgbEvaluator	颜色，如 backgroundColor、textColor 等
ofObject	RectEvaluator	裁剪范围，如 clipBounds

下面是在属性动画中运用插值器和估值器的代码：

```
public class InterpolatorActivity extends AppCompatActivity implements AnimatorListener {
    private TextView tv_interpolator;
```

```
private ObjectAnimator animAcce, animDece, animLinear, animBounce;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_interpolator);
    tv_interpolator = (TextView) findViewById(R.id.tv_interpolator);
    initAnimator();
    initInterpolatorSpinner();
}

private void initInterpolatorSpinner() {
    ArrayAdapter<String> interpolatorAdapter = new ArrayAdapter<String>(this,
            R.layout.item_select, interpolatorArray);
    Spinner sp_interpolator = (Spinner) findViewById(R.id.sp_interpolator);
    sp_interpolator.setPrompt("请选择插值器类型");
    sp_interpolator.setAdapter(interpolatorAdapter);
    sp_interpolator.setOnItemSelectedListener(new InterpolatorSelectedListener());
    sp_interpolator.setSelection(0);
}

private String[] interpolatorArray={
        "背景色+加速插值器+颜色估值器", "旋转+减速插值器+浮点型估值器",
        "裁剪+匀速插值器+矩形估值器", "文字大小+震荡插值器+浮点型估值器"};
class InterpolatorSelectedListener implements OnItemSelectedListener {
    public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
        showInterpolator(arg2);
    }

    public void onNothingSelected(AdapterView<?> arg0) {
    }
}

private void initAnimator() {
    animAcce = ObjectAnimator.ofArgb(tv_interpolator, "backgroundColor", Color.RED, Color.LTGRAY);
    animAcce.setInterpolator(new AccelerateInterpolator());
    animAcce.setEvaluator(new ArgbEvaluator());

    animDece = ObjectAnimator.ofFloat(tv_interpolator, "rotation", 0f, 360f);
    animDece.setInterpolator(new DecelerateInterpolator());
    animDece.setEvaluator(new FloatEvaluator());

    animBounce = ObjectAnimator.ofFloat(tv_interpolator, "textSize", 20f, 60f);
    animBounce.setInterpolator(new BounceInterpolator());
    animBounce.setEvaluator(new FloatEvaluator());
}
```

```
    @TargetApi(Build.VERSION_CODES.JELLY_BEAN_MR2)
    private void showInterpolator(int type) {
        ObjectAnimator anim = null;
        if (type == 0) {
            anim = animAcce;
        } else if (type == 1) {
            anim = animDece;
        } else if (type == 2) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2) {
                int width = tv_interpolator.getWidth();
                int height = tv_interpolator.getHeight();
                animLinear = ObjectAnimator.ofObject(tv_interpolator, "clipBounds",
                        new RectEvaluator(), new Rect(0,0,width,height),
                        new Rect(width/3,height/3,width/3*2,height/3*2),
                        new Rect(0,0,width,height));
                animLinear.setInterpolator(new LinearInterpolator());
                anim = animLinear;
            } else {
                Toast.makeText(this,
                        "矩形估值器要求 Android 为 4.3 以上版本", Toast.LENGTH_SHORT).show();
                return;
            }
        } else if (type == 3) {
            anim = animBounce;
        }
        if (anim != null) {
            anim.setDuration(2000);
            anim.start();
        }
    }
}
```

插值器和估值器的演示效果如图 12-26 和图 12-27 所示。其中，图 12-26 所示为文字大小变大时的画面，图 12-27 所示为文字大小变小时的画面。此处采用的是震荡插值器，由于截图无法准确反映震荡的动画效果，因此建议读者自行编译并运行测试代码，这样会有更直观的感受。

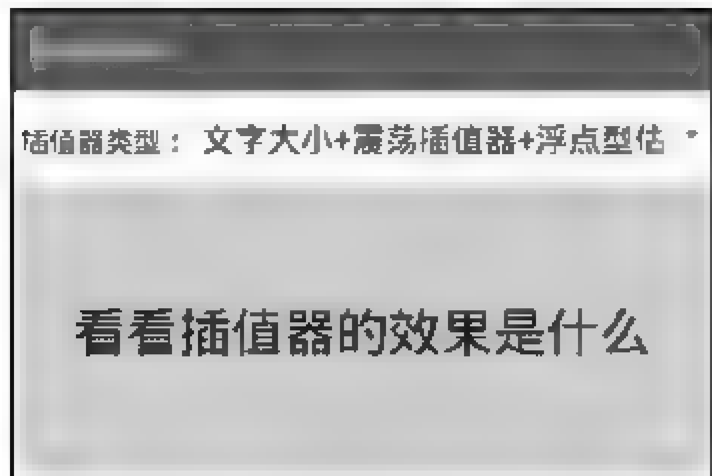


图 12-26　震荡插值器开始播放

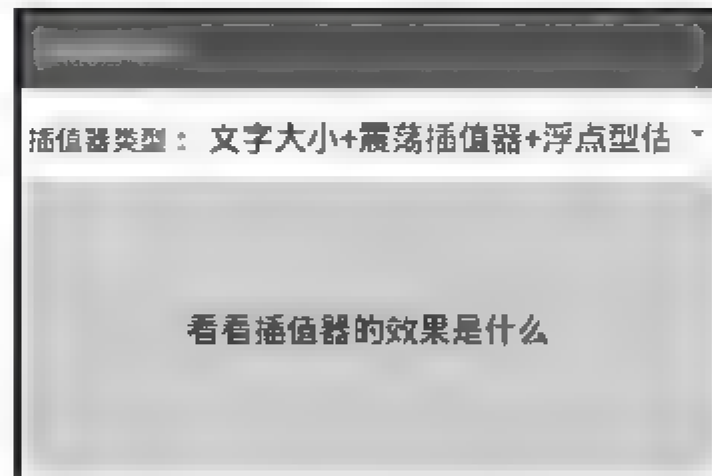


图 12-27　震荡插值器即将结束

12.4 动画的实现手段

本节介绍动画技术常见的 3 种实现手段，包括以帧动画为代表的延时重绘方式、以补间动画和属性动画为代表的设置状态参数方式以及为解决拖曳卡顿问题而采用的滚动器。

12.4.1 使用延时重绘

延时重绘是最基本的动画实现手段，代表技术为帧动画，每隔若干毫秒就用新图片换掉原图片，人眼看过去仿佛画面动起来了。

当然，除了帧动画，还有不少地方采用延时重绘技术，比如第 6 章的圆弧进度动画、第 7 章的 Banner 指示器等，它们都是连续调用 onDraw 或 dispatchDraw 方法实现动画效果。尽管这方面读者已经比较熟悉，不过为加深对该手段的理解，不妨再动手实现一个饼图动画。

下面是饼图动画的参考代码片段：

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    if (mRunning == true) {
        int width = getMeasuredWidth();
        int height = getMeasuredHeight();
        int diameter = Math.min(width, height);
        RectF rectf = new RectF((width - diameter) / 2, (height - diameter) / 2,
                (width + diameter) / 2, (height + diameter) / 2);
        canvas.drawArc(rectf, 0, mDrawingAngle, true, mPaint);
    }
}

private Runnable mRefresh = new Runnable() {
    @Override
    public void run() {
        mDrawingAngle += mIncrease;
        if (mDrawingAngle <= mEndAngle) {
            postInvalidate();
            mHandler.postDelayed(this, mInterval);
        } else {
            mRunning = false;
        }
    }
};
```

饼图动画的播放效果如图 12-28 和图 12-29 所示。其中，图 12-28 所示为饼图动画开始播放时的画面，图 12-29 所示为饼图动画即将结束时的画面。

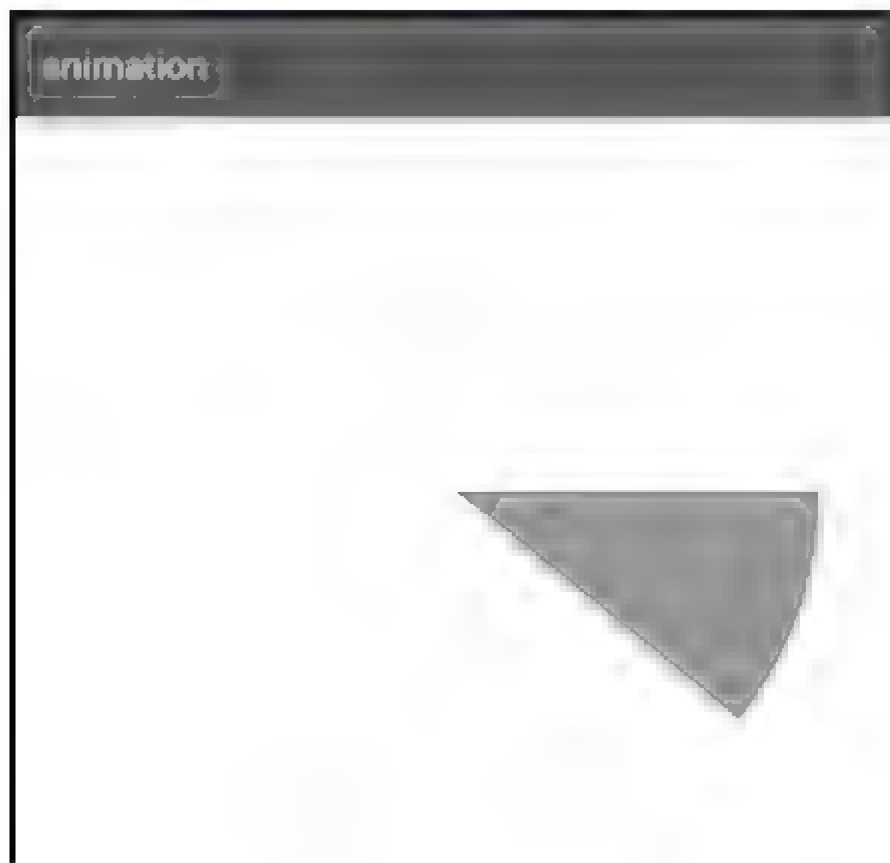


图 12-28　饼图动画开始播放

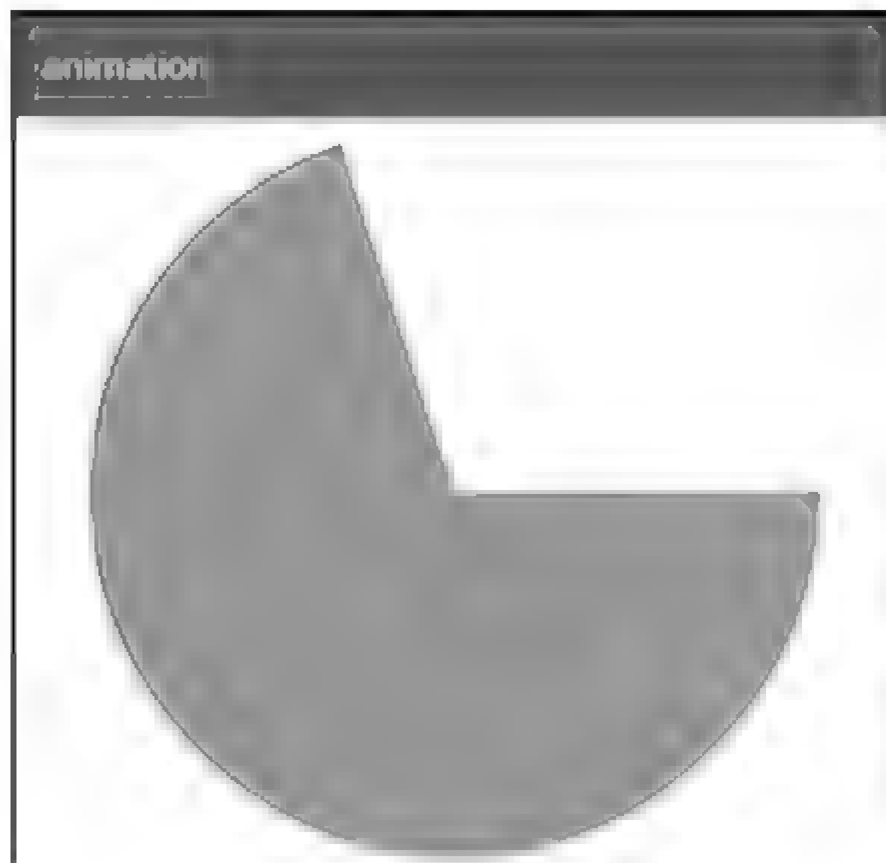


图 12-29　饼图动画即将结束

12.4.2　设置状态参数

设置状态参数是最常见的动画实现手段，代表技术为补间动画和属性动画，通过持续改变视图的状态属性数值让该视图蹦起来、跳起来。

虽然通过属性动画可实现大多数状态变更动画，但是属性动画要求有明确的初始状态值和结束状态值，如果这些起止状态值无法确定，中间还要加入其他运算，属性动画就无法胜任如此复杂的要求，只能自己实现状态变更动画了。

举个例子，经常看朋友圈动态，有的动态内容较多只展示前面一段，如果用户想看完整的需要点击展开动态，看完后再点击收缩动态。这样整个页面的动态列表就会比较均衡，不会出现个别动态占用大片屏幕的情况。查看博客的文章列表也一样，一开始只展示文章开头的几行内容，有需要时再点击显示全篇文章。

点击展开动态，再点击收缩动态，展开与收缩动画其实是不停地变更视图高度。如果动态内容初始展示 3 行文字，初始高度就是每行文字的高度乘以 3，展开后的高度就是每行高度乘以总行数。有了视图高度的起始值和终止值就可以实现动画效果了。

下面是展开动画的参考代码片段：

```
@Override
public void onClick(View v) {
    if (v.getId() == R.id.ll_content) {
        bSelected = !bSelected;
        tv_content.clearAnimation();
        final int deltaValue;
        final int startValue = tv_content.getHeight();
        if (bSelected) {
            deltaValue = tv_content.getLineHeight() * tv_content.getLineCount() - startValue;
        } else {
            deltaValue = tv_content.getLineHeight() * mNormalLines - startValue;
        }
```

```
                Animation animation = new Animation() {
                    protected void applyTransformation(float interpolatedTime, Transformation t) {
                        tv_content.setHeight((int) (startValue + deltaValue * interpolatedTime));
                    }
                };
                animation.setDuration(500);
                tv_content.startAnimation(animation);
            }
        }
```

展开动画的播放效果如图 12-30 和图 12-31 所示。其中，图 12-30 所示为点击文本区域准备播放展开动画时的画面，图 12-31 所示为展开动画即将结束时的画面。

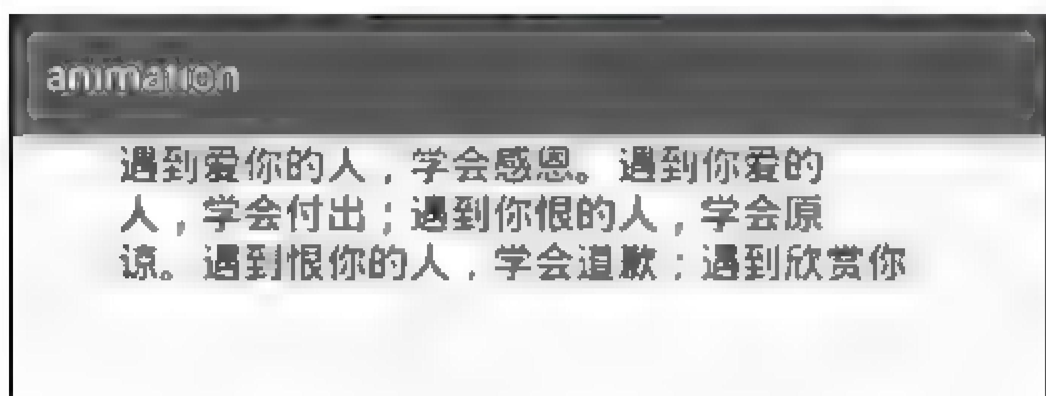


图 12-30　展开动画准备播放

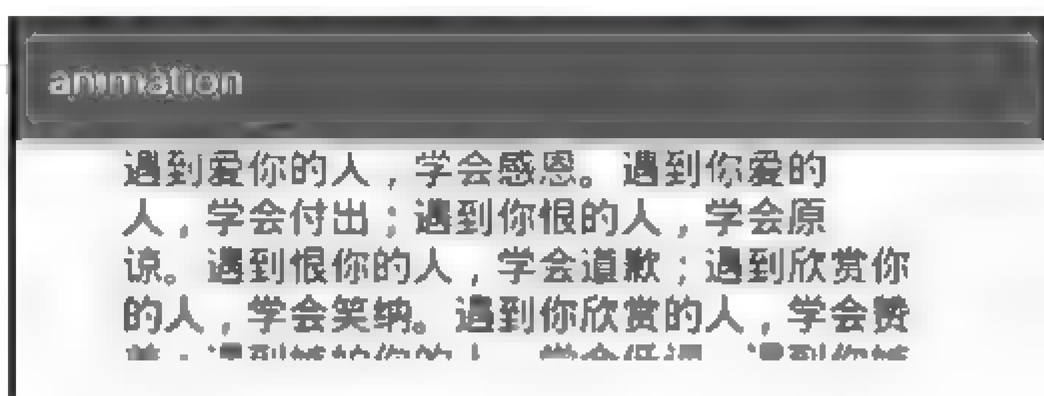


图 12-31　展开动画即将结束

12.4.3　滚动器 Scroller

第 11 章的实战项目通过移动手势拖曳图片到指定位置，拖曳后直接在新位置重绘整个图片，不知道读者有没有发现，这种拖曳方式的画面存在卡顿现象。因为根据人眼的机理，每秒连续播放 20 帧图片才不易感觉到画面卡顿，而拖曳重绘的做法频率绝对小于每秒 20 次，所以自然会出现画面卡顿。

为解决拖曳卡顿的问题，Android 提供了滚动器 Scroller，通过 Scroller 可以实现平滑滚动的效果。下面是 Scroller 的常用方法说明。

- startScroll：设置开始滑动的参数，包括起始的横纵坐标、横纵偏移量和滑动的持续时间。
- computeScrollOffset：计算滑动偏移量。返回值可判断滑动是否结束，返回 fasle 表示滑动结束，返回 true 表示还在滑动中。
- getCurrX：获得当前的横坐标。
- getCurrY：获得当前的纵坐标。
- getDuration：获得滑动的持续时间。
- forceFinished：强行停止滑动。
- isFinished：判断滑动是否结束。返回 fasle 表示还未结束，返回 true 表示滑动结束。

该方法与 computeScrollOffset 的区别在于：

（1）computeScrollOffset 内部计算偏移量，而 isFinished 只返回标志不做其他处理。
（2）computeScrollOffset 返回 fasle 表示滑动结束，而 isFinished 返回 true 表示滑动结束。

虽然滚动器提供了滑动的相关计算函数，但是并不能直接滑动视图。因为 Scroller 是一个运算模拟器，根据时间的流逝计算横纵坐标偏移，要想让视图真正动起来，还得调用视图自身的滑动方法处理滑动操作，即调用 scrollTo 和 scrollBy 两个方法。

- scrollTo：将视图滑动到指定坐标位置。
- scrollBy：将视图滑动指定偏移量。查看源码会发现 scrollBy 方法内部就是调用 scrollTo 方法，当然得先给当前坐标加上偏移量，从而得到滑动后的绝对坐标。

下面是使用滚动器的参考代码：

```
public class ScrollTextView extends TextView {
    private Scroller mScroller;

    public ScrollTextView(Context context) {
        this(context, null);
    }

    public ScrollTextView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mScroller = new Scroller(context);
    }

    public void smoothScrollTo(int fx, int fy) {
        int dx = fx - mScroller.getFinalX();
        int dy = fy - mScroller.getFinalY();
        smoothScrollBy(dx, dy);
    }

    public void smoothScrollBy(int dx, int dy) {
        // 设置滚动偏移量，注意正数是往左滚、往上滚，负数才是往右滚、往下滚
        mScroller.startScroll(mScroller.getFinalX(), mScroller.getFinalY(), -dx, -dy);
        // 调用 invalidate 方法才能保证 computeScroll 方法被调用
        invalidate();
    }

    @Override
    public void computeScroll() {
        if (mScroller.computeScrollOffset()) {  // 先判断 mScroller 滚动是否完成
            scrollTo(mScroller.getCurrX(), mScroller.getCurrY());  //调用 scrollTo 方法完成实际滚动
            postInvalidate();  // 刷新页面
        }
        super.computeScroll();
    }
}
```

滚动器的演示效果如图 12-32 和图 12-33 所示。其中，图 12-32 所示为视图滚动开始前的画面，图 12-33 所示为视图滚动结束后的画面。单看截图不方便观察动画效果，建议读者自己运行代码查看效果图。



图 12-32　视图尚未开始滚动



图 12-33　视图滚动已经结束

12.5　实战项目：仿 QQ 空间的动感影集

动画可以做得千变万化、很酷很炫，故而常用于展示具有纪念意义的组图，比如婚纱照、亲子照、艺术照等。这方面做得比较好、使用比较广泛的当数 QQ 空间的动感影集，用户添加一组图片，动感影集便给每张图片渲染不同的动画效果，让原本静止的图片变得活泼起来，辅以各种精致的动画特效，营造一种赏心悦目的感觉。本章以“仿 QQ 空间的动感影集”为实战项目，结合本章的动画技术实现开发者自己的动感影集。

12.5.1　设计思路

动感影集的目的是使用动画技术呈现前后图片的动态切换效果，用到的动画必须承上启下，而且要求具备一定的视觉美感。以这样的标准来衡量，目前适用于动感影集的动画种类不算多，下面都拿来练练手。动感影集的播放效果如图 12-34 所示，很明显这是一个包含旋转动画在内的集合动画。



图 12-34　动感影集中的集合动画效果

当然，实战项目的动感影集不仅采用集合动画，还包括其他种类动画，读者不妨先列举一部分，看看有哪些能够应用在动感影集中。下面是笔者罗列的部分影集动画技术。

（1）淡入淡出动画：用于前后两张图片的渐变切换。
（2）灰度动画：用于从无到有渐变显示一张图片。
（3）平移动画：用于把上层图片抽离当前视图。
（4）缩放动画：用于逐步缩小并隐没上层图片。
（5）旋转动画：用于将上层图片甩离当前视图。
（6）裁剪动画：用于把上层图片逐步裁剪完。
（7）其余动画：更多动画特效切换，包括百叶窗动画、马赛克动画等。

动画技术用起来不难，关键要用好，只有用到位才能让你的 App 熠熠生辉、锦上添花。

12.5.2 小知识：画布的绘图层次

本书到目前为止，画布 Canvas 上的绘图操作都是在同一个图层上进行的。这就意味着如果存在重叠区域，后面绘制的图形就必然覆盖前面的图形。但绘图是比较复杂的事情，不是直接覆盖这么简单，有些特殊的绘图操作往往需要做与、或、非运算，如此才能实现百变的图像特效。

Android 给画布的图层显示制定了许多规则，详细的图层显示规则见表 12-7。表中的上层指的是后面绘制的图形 Src，下层指的是前面绘制的图形 Dst。

表12-7 图层模式的取值说明

PorterDuff.Mode 类的图层模式	说明
CLEAR	不显示任何图形
SRC	只显示上层图形
DST	只显示下层图形
SRC_OVER	按通常情况显示，即重叠部分由上层遮盖下层
DST_OVER	重叠部分由下层遮盖上层，其余部分正常显示
SRC_IN	只显示重叠部分的上层图形
DST_IN	只显示重叠部分的下层图形
SRC_OUT	只显示上层图形的未重叠部分
DST_OUT	只显示下层图形的未重叠部分
SRC_ATOP	只显示上层图形区域，但重叠部分显示下层图形
DST_ATOP	只显示下层图形区域，但重叠部分显示上层图形
XOR	不显示重叠部分，其余部分正常显示
DARKEN	重叠部分按颜料混合方式加深，其余部分正常显示
LIGHTEN	重叠部分按光照重合方式加亮，其余部分正常显示
MULTIPLY	只显示重叠部分，且重叠部分的颜色混合加深
SCREEN	过滤重叠部分的深色，其余部分正常显示

这些图层规则的文字说明有点令人费解，还是看画面效果比较直观。如图 12-35 所示，圆圈是先绘制的图形，正方形是后绘制的图形，图例展示了运用不同规则时的显示画面。合理运用图层规则可以实现酷炫的动画效果，比如百叶窗动画、马赛克动画等。

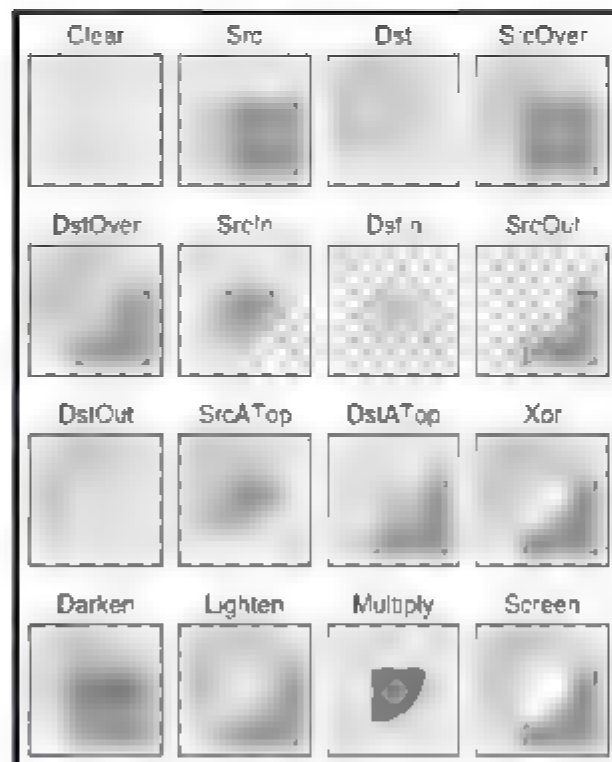


图 12-35 各种图层规则的画面效果

要想在画布中使用图层规则，就要调用画布对象的 setXfermode 方法，并指定相应的图层模式。下面是百叶窗视图的代码，其中采用了 DST_IN 模式：



```
public class ShutterView extends View {
    private final static String TAG = "ShutterView";
    private Context mContext;
    private Paint mPaint;
    private int mOriention = LinearLayout.HORIZONTAL;
    private int mLeafCount = 10;
    private PorterDuff.Mode mMode = PorterDuff.Mode.DST_IN;
    private Bitmap mBitmap;
    private int mRatio = 0;

    public ShutterView(Context context) {
        this(context, null);
    }

    public ShutterView(Context context, AttributeSet attrs) {
        super(context, attrs);
        mContext = context;
        mPaint = new Paint();
    }

    public void setOriention(int oriention) {
        mOriention = oriention;
    }

    public void setLeafCount(int leaf_count) {
        mLeafCount = leaf_count;
    }

    public void setMode(PorterDuff.Mode mode) {
        mMode = mode;
    }

    public void setImageBitmap(Bitmap bitmap) {
        mBitmap = bitmap;
    }

    public void setRatio(int ratio) {
        mRatio = ratio;
        invalidate();
    }

    @SuppressLint("DrawAllocation")
    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        if (mBitmap == null) {
```

```
            return;
        }
        int width = getMeasuredWidth();
        int height = getMeasuredHeight();
        canvas.drawColor(Color.TRANSPARENT);
        // 创建图片的遮罩
        Bitmap mask = Bitmap.createBitmap(width, height, mBitmap.getConfig());
        Canvas canvasMask = new Canvas(mask);
        for (int i = 0; i < mLeafCount; i++) {
            if (mOrientation == LinearLayout.HORIZONTAL) {
                int column_width = (int) Math.ceil(width*1f / mLeafCount);
                int left = column_width * i;
                int right = left + column_width * mRatio / 100;
                canvasMask.drawRect(left, 0, right, height, mPaint);
            } else {
                int row_height = (int) Math.ceil(height*1f / mLeafCount);
                int top = row_height * i;
                int bottom = top + row_height * mRatio / 100;
                canvasMask.drawRect(0, top, width, bottom, mPaint);
            }
        }
        // 设置离屏缓存
        int saveLayer = canvas.saveLayer(0, 0, width, height, null, Canvas.ALL_SAVE_FLAG);
        // 绘制目标图像
        Rect src = new Rect(0, 0, mBitmap.getWidth(), mBitmap.getHeight());
        Rect dst = new Rect(0, 0, width, width*mBitmap.getHeight()/mBitmap.getWidth());
        canvas.drawBitmap(mBitmap, src, dst, mPaint);
        // 设置混合模式 （只在源图像和目标图像相交的地方绘制目标图像）
        mPaint.setXfermode(new PorterDuffXfermode(mMode));
        // 再绘制 src 源图
        canvas.drawBitmap(mask, 0, 0, mPaint);
        // 还原混合模式
        mPaint.setXfermode(null);
        // 还原画布
        canvas.restoreToCount(saveLayer);
    }
}
```

百叶窗视图 ShutterView 仅仅是一个静态画面，若想让它动起来形成百叶窗动画还得利用属性动画渐进设置 ratio 属性，使整个百叶窗的各个叶片逐步合上，从而实现动画特效。下面是百叶窗动画的代码：

```
public class ShutterActivity extends AppCompatActivity {
    private ShutterView sv_shutter;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_shutter);
        sv_shutter = (ShutterView) findViewById(R.id.sv_shutter);
        sv_shutter.setImageBitmap(BitmapFactory.decodeResource(getResources(), R.drawable.bdg03));
        initShutterSpinner();
    }

    private void initShutterSpinner() {
        ArrayAdapter<String> shutterAdapter = new ArrayAdapter<String>(this,
                R.layout.item_select, shutterArray);
        Spinner sp_shutter = (Spinner) findViewById(R.id.sp_shutter);
        sp_shutter.setPrompt("请选择百叶窗动画类型");
        sp_shutter.setAdapter(shutterAdapter);
        sp_shutter.setOnItemSelectedListener(new ShutterSelectedListener());
        sp_shutter.setSelection(0);
    }

    private String[] shutterArray={"水平五叶", "水平十叶", "水平二十叶",
            "垂直五叶", "垂直十叶", "垂直二十叶"};
    class ShutterSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            sv_shutter.setOriention((arg2<3)?LinearLayout.HORIZONTAL:LinearLayout.VERTICAL);
            if (arg2 == 0 || arg2 == 3) {
                sv_shutter.setLeafCount(5);
            } else if (arg2 == 1 || arg2 == 4) {
                sv_shutter.setLeafCount(10);
            } else if (arg2 == 2 || arg2 == 5) {
                sv_shutter.setLeafCount(20);
            }
            ObjectAnimator anim = ObjectAnimator.ofInt(sv_shutter, "ratio", 0, 100);
            anim.setDuration(3000);
            anim.start();
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }
}
```

百叶窗动画的播放效果如图 12-36 和图 12-37 所示。其中，图 12-36 所示为百叶窗动画开始播放时的画面，图 12-37 所示为百叶窗动画即将结束播放时的画面。





图 12-36　百叶窗动画开始播放



图 12-37　百叶窗动画即将结束播放

马赛克动画的实现原理与百叶窗动画类似，只是在绘制图片遮罩时选择了不同的算法，其余步骤与百叶窗动画是一样的。马赛克视图的 MosaicView 代码参见本书的下载资源，马赛克动画的播放效果如图 12-38 和图 12-39 所示。其中，图 12-38 所示为马赛克动画开始播放时的画面，图 12-39 所示为马赛克动画即将结束播放时的画面。



图 12-38　马赛克动画开始播放



图 12-39　马赛克动画即将结束

12.5.3　代码示例

编码与测试方面需要注意以下两点：

（1）如果把动画描述定义在 XML 文件中，注意动画定义文件要放在 res/anim 目录下。

（2）测试手机的 Android 版本要求不低于 Android 4.3，因为裁剪动画用到的矩形估值器 RectEvaluator 是在 Android 4.3 之后引入的。

动感影集的测试挺简单的，无须什么操作流程，只要静静欣赏屏幕上的动画轮播即可。动感影集的轮播效果如图 12-40～图 12-45 所示。其中，图 12-40 展示了灰度动画，图 12-41 展示了裁剪动画，图 12-42 展示了百叶窗动画，图 12-43 展示了马赛克动画，图 12-44 展示了淡入淡出动画，图 12-45 展示了平移动画。



图 12-40　动感影集的灰度动画效果



图 12-41　动感影集的裁剪动画效果



图 12-42　动感影集的百叶窗动画效果



图 12-43　动感影集的马赛克动画效果



图 12-44　动感影集的淡入淡出动画效果



图 12-45　动感影集的平移动画效果

为方便演示，动感影集未做成可选择图片文件的形式，而是在代码中固定了几张演示图片。另外，各种动画的执行顺序也是固定的，没有做成可定制动画顺序。读者若有兴趣，可在源码的基础上进行改造，使其更贴近真实动感影集使用习惯。

下面是动感影集的活动页面代码：

```
public class YingjiActivity extends AppCompatActivity implements
            AnimatorListener, AnimationListener, OnClickListener {
    private RelativeLayout rl_yingji;
    private TextView tv_anim_title;
    private LayoutParams mParams;
    private ImageView view1, view4, view5, view6;
    private ShutterView view2;
    private MosaicView view3;
    private int[] mImageArray = {
                R.drawable.bdg01, R.drawable.bdg02, R.drawable.bdg03, R.drawable.bdg04, R.drawable.bdg05,
                R.drawable.bdg06, R.drawable.bdg07, R.drawable.bdg08, R.drawable.bdg09, R.drawable.bdg10
    };
    private ObjectAnimator anim1, anim2, anim3, anim4;
    private Animation translateAnim, setAnim;
    private int mDuration = 5000;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_yingji);
        rl_yingji = (RelativeLayout) findViewById(R.id.rl_yingji);
        tv_anim_title = (TextView) findViewById(R.id.tv_anim_title);
        playYingji();
    }

    private void initView() {
        mParams = new LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT);
        view1 = new ImageView(this);   //灰度动画和裁剪动画
        view1.setLayoutParams(mParams);
        view1.setImageResource(mImageArray[0]);
        view1.setScaleType(ScaleType.FIT_START);
        view1.setAlpha(0f);   //百叶窗动画
        view2 = new ShutterView(this);
        view2.setLayoutParams(mParams);
        view2.setImageBitmap(BitmapFactory.decodeResource(getResources(), mImageArray[1]));
        view2.setMode(PorterDuff.Mode.DST_OUT);
        view3 = new MosaicView(this);   //马赛克动画
        view3.setLayoutParams(mParams);
        view3.setImageBitmap(BitmapFactory.decodeResource(getResources(), mImageArray[2]));
        view3.setMode(PorterDuff.Mode.DST_OUT);
        view3.setRatio(-5);
        view4 = new ImageView(this);   //淡入淡出动画
```

```
        view4.setLayoutParams(mParams);
        view4.setImageResource(mImageArray[3]);
        view4.setScaleType(ScaleType.FIT_START);
        view5 = new ImageView(this);  //平移动画
        view5.setLayoutParams(mParams);
        view5.setImageResource(mImageArray[5]);
        view5.setScaleType(ScaleType.FIT_START);
        view6 = new ImageView(this);  //集合动画
        view6.setLayoutParams(mParams);
        view6.setImageResource(mImageArray[6]);
        view6.setScaleType(ScaleType.FIT_START);
    }

    private void playYingji() {
        rl_yingji.removeAllViews();
        initView();
        rl_yingji.addView(view1);
        anim1 = ObjectAnimator.ofFloat(view1, "alpha", 0f, 1f);
        anim1.setDuration(mDuration);
        anim1.addListener(this);
        anim1.start();
    }

    @Override
    public void onAnimationStart(Animator animation) {
        if (animation.equals(anim1)) {
            tv_anim_title.setText("正在播放灰度动画");
        } else if (animation.equals(anim2)) {
            tv_anim_title.setText("正在播放裁剪动画");
        } else if (animation.equals(anim3)) {
            tv_anim_title.setText("正在播放百叶窗动画");
        } else if (animation.equals(anim4)) {
            tv_anim_title.setText("正在播放马赛克动画");
        }
    }

    @TargetApi(Build.VERSION_CODES.JELLY_BEAN_MR2)
    @Override
    public void onAnimationEnd(Animator animation) {
        if (animation.equals(anim1)) {
            rl_yingji.addView(view2, 0);
            Bitmap bitmap = BitmapFactory.decodeResource(getResources(), mImageArray[0]);
            int width = view1.getWidth();
            int height = bitmap.getHeight()*width/bitmap.getWidth();
```

```
            anim2 = ObjectAnimator.ofObject(view1, "clipBounds",
                    new RectEvaluator(), new Rect(0,0,width,height),
                    new Rect(width/2,height/2,width/2,height/2));
            anim2.setDuration(mDuration);
            anim2.addListener(this);
            anim2.start();
        } else if (animation.equals(anim2)) {
            rl_yingji.removeView(view1);
            rl_yingji.addView(view3, 0);
            anim3 = ObjectAnimator.ofInt(view2, "ratio", 0, 100);
            anim3.setDuration(mDuration);
            anim3.addListener(this);
            anim3.start();
        } else if (animation.equals(anim3)) {
            rl_yingji.removeView(view2);
            rl_yingji.addView(view4, 0);
            int offset = 5;
            view3.setOffset(offset);
            anim4 = ObjectAnimator.ofInt(view3, "ratio", 0-offset, 101+offset);
            anim4.setDuration(mDuration);
            anim4.addListener(this);
            anim4.start();
        } else if (animation.equals(anim4)) {
            rl_yingji.removeView(view3);
            Drawable[] drawableArray = { getResources().getDrawable(mImageArray[3]),
                    getResources().getDrawable(mImageArray[4]) };
            TransitionDrawable td_fade = new TransitionDrawable(drawableArray);
            td_fade.setCrossFadeEnabled(false);
            view4.setImageDrawable(td_fade);
            int delay = mDuration;
            td_fade.startTransition(delay);
            tv_anim_title.setText("正在播放淡入淡出动画");
            mHandler.postDelayed(mTransitionEnd, delay);
        }
    }

    private Handler mHandler = new Handler();
    private Runnable mTransitionEnd = new Runnable() {
        @Override
        public void run() {
            rl_yingji.addView(view5, 0);
            translateAnim = new TranslateAnimation(0f, -view4.getWidth(), 0f, 0f);
            translateAnim.setDuration(mDuration);
```



```
            translateAnim.setFillAfter(true);
            view4.startAnimation(translateAnim);
            translateAnim.setAnimationListener(YingjiActivity.this);
        }
    };

    private void startSetAnim() {
        Animation alpha = new AlphaAnimation(1.0f, 0.1f);
        alpha.setDuration(mDuration);
        alpha.setFillAfter(true);
        Animation translate = new TranslateAnimation(1.0f, -200f, 1.0f, 1.0f);
        translate.setDuration(mDuration);
        translate.setFillAfter(true);
        Animation scale = new ScaleAnimation(1.0f, 1.0f, 1.0f, 0.5f);
        scale.setDuration(mDuration);
        scale.setFillAfter(true);
        Animation rotate = new RotateAnimation(0f,    360f, Animation.RELATIVE_TO_SELF,
                0.5f, Animation.RELATIVE_TO_SELF, 0.5f);
        rotate.setDuration(mDuration);
        rotate.setFillAfter(true);
        setAnim = new AnimationSet(true);
        ((AnimationSet) setAnim).addAnimation(alpha);
        ((AnimationSet) setAnim).addAnimation(translate);
        ((AnimationSet) setAnim).addAnimation(scale);
        ((AnimationSet) setAnim).addAnimation(rotate);
        setAnim.setFillAfter(true);
        view5.startAnimation(setAnim);
        setAnim.setAnimationListener(this);
    }

    @Override
    public void onAnimationCancel(Animator animation) {
    }

    @Override
    public void onAnimationRepeat(Animator animation) {
    }

    @Override
    public void onAnimationStart(Animation animation) {
        if (animation.equals(translateAnim)) {
            tv_anim_title.setText("正在播放平移动画");
        } else if (animation.equals(setAnim)) {
            tv_anim_title.setText("正在播放集合动画");
        }
```

```
        }

        @Override
        public void onAnimationEnd(Animation animation) {
            if (animation.equals(translateAnim)) {
                rl_yingji.removeView(view4);
                rl_yingji.addView(view6, 0);
                startSetAnim();
            } else if (animation.equals(setAnim)) {
                rl_yingji.removeView(view5);
                tv_anim_title.setText("动感影集播放结束，谢谢观看");
                view6.setOnClickListener(this);
            }
        }

        @Override
        public void onAnimationRepeat(Animation animation) {
        }

        @Override
        public void onClick(View v) {
            if (v.equals(view6)) {
                playYingji();
            }
        }
    }
```

12.6 小　　结

本章主要介绍了 App 开发用到的常见动画技术，包括帧动画的用法（帧动画、GIF 动画、淡入淡出动画）、补间动画的用法（补间动画的种类与用法、集合动画、在飞掠横幅中使用补间动画）、属性动画的用法（属性动画、属性动画组合、插值器和估值器）、常见的动画实现手段（使用延时重绘、设置状态参数、滚动器 Scroller）。最后设计了一个实战项目“仿 QQ 空间的动感影集”，在该项目的 App 编码中采用本章介绍的主要动画技术，实现了图片动态轮换的效果。另外。介绍了画布的绘图层次，以及如何实现百叶窗动画和马赛克动画。

通过本章的学习，读者应该能够掌握以下 4 种开发技能：

（1）学会如何使用帧动画实现动态效果。

（2）学会在合适的场合使用补间动画。

（3）学会属性动画的基本用法和高级用法。

（4）学会常用的几种动画实现手段。





多　媒　体

本章介绍 App 开发常见的多媒体技术，主要包括如何使用各种图像控件实现自定义相册、如何使用视频相关控件实现视频播放器，另外介绍四大组件之一的ContentProvider的基本概念与常见用法。最后结合本章所学的知识演示一个实战项目“音乐播放器——浪花音乐”的设计与实现。

13.1 相　　册

本节介绍自定义相册的实现过程，首先说明使用画廊或循环视图如何实现简单相册；接着阐述使用图像切换器如何实现相册的左右滑动功能；然后分别介绍卡片视图与调色板的用法，并结合上述图像控件完成一个图片查看器——青青相册。

13.1.1 画廊 Gallery

前几章使用文件对话框打开图片时只能看到图片的文件名，看不到图片的缩略图，对用户来说很不方便，因为光看文件名怎么知道这张图片什么模样呢？如果是在电脑上，就可以查看一组图片的缩略图列表，很容易找到想要的图片。在手机上可以使用相应的图像控件做出缩略图展示的相册效果。

画廊 Gallery 是专门用于展示图片列表的控件，左右滑动手势即可展示内嵌的图片列表，画面效果类似于一个平面万花筒。尽管 Android 将 Gallery 标记为 Deprecation（表示已废弃），建议开发者采用 HorizontalScrollView 或 ViewPager 代替，不过 Gallery 用来轮播图片是一个挺好的选择。不妨了解一下 Gallery 控件，并结合其他控件加深对图像开发的理解。

下面是 Gallery 的常用方法说明。

- setSpacing：设置图片之间的间隔大小，对应的 XML 属性是 spacing。
- setUnselectedAlpha：设置未选定图片的透明度，对应的 XML 属性是 unselectedAlpha。取值范围为 0.0～1.0，0.0 表示完全透明，1.0 表示完全不透明。
- setAdapter：设置画廊的适配器。
- getSelectedItemId：获取当前选中的视图序号。
- setSelection：设置当前选中第几个视图。
- setOnItemClickListener：设置单项的点击监听器。

使用画廊看起来很简单，接下来试着用 Gallery 结合 ImageView 实现观看画廊的相册效果。首先在布局文件中放置一个框架布局 FrameLayout，里面放一个画廊控件与一个图像视图控件，ImageView 设置为充满整个屏幕，Gallery 放在屏幕下方；然后监听 Gallery 控件的单项点击事件，当用户点击指定图片项时，使用 ImageView 控件填充该图片，也就是点小图看大图。

下面是通过 Gallery 与 ImageView 实现简单相册的代码：

```
public class GalleryActivity extends AppCompatActivity implements OnItemClickListener {
    private ImageView iv_gallery;
    private Gallery gl_gallery;
    private int[] mImageRes = { R.drawable.scene1, R.drawable.scene2, R.drawable.scene3,
            R.drawable.scene4, R.drawable.scene5, R.drawable.scene6 };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_gallery);
        iv_gallery = (ImageView) findViewById(R.id.iv_gallery);
        iv_gallery.setImageResource(mImageRes[0]);
        int dip_pad = Utils.dip2px(this, 20);
        gl_gallery = (Gallery) findViewById(R.id.gl_gallery);
        gl_gallery.setPadding(0, dip_pad, 0, dip_pad);
        gl_gallery.setSpacing(dip_pad);
        gl_gallery.setUnselectedAlpha(0.5f);
        gl_gallery.setAdapter(new GalleryAdapter(this, mImageRes));
        gl_gallery.setOnItemClickListener(this);
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        iv_gallery.setImageResource(mImageRes[position]);
    }
}
```

Gallery 相册的画面效果如图 13-1 和图 13-2 所示。其中，图 13-1 所示为展示相册第一张图片时的画面；图 13-2 所示为点击第二张小图时，屏幕展示第二张大图的画面。



图 13-1 画廊展示第一张图片



图 13-2 画廊展示第二张图片

如果想用其他控件替代 Gallery，就可以考虑使用功能强大的循环视图 RecyclerView。具体实现时主要是定义一个水平方向的线性布局管理器，然后通过适配器填入图片列表。

使用 RecyclerView 与 ImageView 实现相册的代码很简单，举例如下：

```
public class RecyclerViewActivity extends AppCompatActivity implements OnItemClickListener {
    private ImageView iv_photo;
    private RecyclerView rv_photo;
    private int[] mImageRes = { R.drawable.scene1, R.drawable.scene2, R.drawable.scene3,
            R.drawable.scene4, R.drawable.scene5, R.drawable.scene6 };
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_recycler_view);
        iv_photo = (ImageView) findViewById(R.id.iv_photo);
        iv_photo.setImageResource(mImageRes[0]);
        rv_photo = (RecyclerView) findViewById(R.id.rv_photo);
        LinearLayoutManager manager = new LinearLayoutManager(this);
        manager.setOrientation(LinearLayout.HORIZONTAL);
        rv_photo.setLayoutManager(manager);
        PhotoAdapter adapter = new PhotoAdapter(this, mImageRes);
        adapter.setOnItemClickListener(this);
        rv_photo.setAdapter(adapter);
        rv_photo.setItemAnimator(new DefaultItemAnimator());
        rv_photo.addItemDecoration(new SpacesItemDecoration(20));
    }

    @Override
    public void onItemClick(View view, int position) {
        iv_photo.setImageResource(mImageRes[position]);
        rv_photo.scrollToPosition(position);
    }
}
```

使用 RecyclerView 方式实现的相册效果如图 13-3 和图 13-4 所示。其中，图 13-3 所示为展示相册第 3 张图片时的画面；图 13-4 所示为点击第 4 张小图时，屏幕展示第 4 张大图的画面。



图 13-3　循环视图展示第 3 张图片



图 13-4　循环视图展示第 4 张图片

13.1.2　图像切换器 ImageSwitcher

可能读者已经发现，前面 Gallery 相册在切换大图时比较生硬，前后两张图片闪一下就切过去了，用户体验不够友好。有没有办法让图片切换自然一些呢，比如通过渐变动画的方式？



答案肯定是有的，就是把占据整个屏幕的图像视图 ImageView 换成图像切换器 ImageSwitcher，然后通过 ImageSwitcher 实现前后图片的切换动画。

ImageSwitcher 继承自视图动画器 ViewAnimator，用于承载前后两个图像的变换动画；与之对应的是，文本切换器 TextSwitcher 承载前后两个文本的变换动画；第 11 章介绍的飞掠视图 ViewFlipper 是从 ViewAnimator 派生而来，读者已经知道它用来承载前后两个视图的变换动画。

下面介绍 ImageSwitcher 的常用方法。

- setFactory：设置一个视图工厂。该视图工厂由 ViewFactory 派生而来，需重写 makeView 方法返回工厂的具体视图。对于 ImageSwitcher 来说，工厂返回的是 ImageView 对象。
- setImageResource：设置当前图像的资源 ID。该方法与下面的 setImageDrawable 方法和 setImageURI 方法为三选一操作，调用了其中一个方法，就无须调用另外两个方法。
- setImageDrawable：设置当前图像的 Drawable 对象。
- setImageURI：设置当前图像的 URI 地址。
- setInAnimation：设置后一个图像的进入动画。
- setOutAnimation：设置前一个图像的退出动画。

这里运用的动画技术跟第 11 章和第 12 章的飞掠视图类似。首先，对前后图片的切换动画可以事先设置好集合动画，通过 setInAnimation 和 setOutAnimation 方法完成动画调用；其次，前后图片的切换操作不但可由 Gallery 控件的点击操作出发，而且可由手势的左滑和右滑操作触发，这要借助于手势检测器 GestureDetector，通过检测左滑手势和右滑手势自动轮播图片。

按照以上的设计思路使用 ImageSwitcher 实现相册切换动画的代码如下：

```
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
    is_switcher.setInAnimation(AnimationUtils.loadAnimation(this, R.anim.fade_in));
    is_switcher.setOutAnimation(AnimationUtils.loadAnimation(this, R.anim.fade_out));
    is_switcher.setImageResource(mImageRes[position]);
}

public class ViewFactoryImpl implements ViewFactory {
    @Override
    public View makeView() {
        ImageView iv = new ImageView(ImageSwitcherActivity.this);
        iv.setBackgroundColor(0xFFFFFFFF);
        iv.setScaleType(ScaleType.FIT_XY);
        iv.setLayoutParams(new ImageSwitcher.LayoutParams(
                LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT));
        return iv;
    }
}

@Override
```

```
public boolean onTouch(View v, MotionEvent event) {
    mGesture.onTouchEvent(event);
    return true;
}

@Override
public void gotoNext() {
    is_switcher.setInAnimation(AnimationUtils.loadAnimation(this, R.anim.push_left_in));
    is_switcher.setOutAnimation(AnimationUtils.loadAnimation(this, R.anim.push_left_out));
    int next_pos = (int) (gl_switcher.getSelectedItemId() + 1);
    if (next_pos >= mImageRes.length) {
        next_pos = 0;
    }
    is_switcher.setImageResource(mImageRes[next_pos]);
    gl_switcher.setSelection(next_pos);
}

@Override
public void gotoPre() {
    is_switcher.setInAnimation(AnimationUtils.loadAnimation(this, R.anim.push_right_in));
    is_switcher.setOutAnimation(AnimationUtils.loadAnimation(this, R.anim.push_right_out));
    int pre_pos = (int) (gl_switcher.getSelectedItemId() - 1);
    if (pre_pos < 0) {
        pre_pos = mImageRes.length - 1;
    }
    is_switcher.setImageResource(mImageRes[pre_pos]);
    gl_switcher.setSelection(pre_pos);
}
```

相册切换动画的效果如图 13-5 和图 13-6 所示。其中，图 13-5 所示为切换开始的画面，此时右边图片缓缓移入屏幕；图 13-6 所示为切换即将结束的画面，此时右边图片已经大部分移入屏幕，左边图片快要移出屏幕了。



图 13-5　图像切换刚刚开始的画面

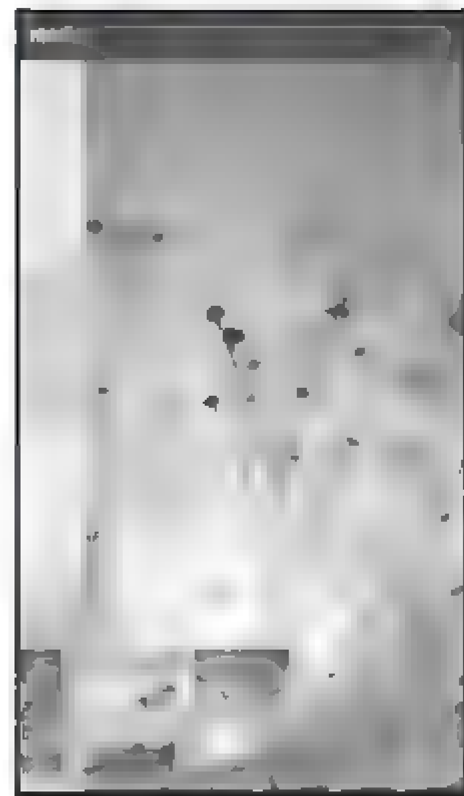


图 13-6　图像切换即将结束的画面

13.1.3 图片查看器——青青相册

经过 Gallery 和 ImageSwitcher 的配合，这个相册有点像模像样了。当然，作为孜孜不倦、勤奋好学的开发者，绝不能满足于一点雕虫小技。接下来我们再加上一些技术，让相册变得更加赏心悦目。

首先加入的技术是卡片视图 CardView，该视图是 Android 在 5.0 后引入的新控件。顾名思义，CardView 拥有一个卡片式的圆角边框，边框外缘有一圈阴影，边框内缘有一圈空白。准确地说，CardView 实际上是一个布局视图，继承自 Framelayout，可以当作具有边框效果的特殊布局。

因为 CardView 是 5.0 之后的新增控件，所以为了兼容以前的 Android 版本，在使用该控件前要先修改 build.gradle，即在 dependencies 节点中加入下面一行代码表示导入 cardview 库：

```
compile 'com.android.support:cardview-v7:25.0.0'
```

CardView 的常用属性与方法的说明见表 13-1。

表13-1　CardView的常用属性与方法说明

CardView 的属性名称	CardView 的设置方法	说明
cardBackgroundColor	setCardBackgroundColor	设置卡片边框的背景颜色
cardCornerRadius	setRadius	设置卡片边框的圆角半径
cardElevation	setCardElevation	设置卡片边缘的阴影高程，即宽度
contentPadding	setContentPadding	设置卡片边框的间隔

使用 CardView 属性时需要注意以下两点：

（1）因为 cardview 库是作为外部库导入的，所以节点属性要像对待自定义控件一样，即先在根节点定义一个命名空间 app 指向 res-auto，然后使用“app:属性名称”的形式定义属性值，不可直接使用“android:属性名称”。

（2）在设置阴影宽度的同时设置对应宽度的 margin，因为阴影宽度不计入卡片的宽高，如果卡片宽高设置为 wrap_content，阴影部分就会被自动截掉。

下面是使用 CardView 的代码：

```
public class CardViewActivity extends AppCompatActivity {
    private CardView cv_card;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_card_view);
        cv_card = (CardView) findViewById(R.id.cv_card);
        initCardSpinner();
    }

    private void initCardSpinner() {
```

```
        ArrayAdapter<String> cardAdapter = new ArrayAdapter<String>(this,
                R.layout.item_select, cardArray);
        Spinner sp_card = (Spinner) findViewById(R.id.sp_card);
        sp_card.setPrompt("请选择卡片视图类型");
        sp_card.setAdapter(cardAdapter);
        sp_card.setOnItemSelectedListener(new CardSelectedListener());
        sp_card.setSelection(0);
    }

    private String[] cardArray = {"圆角与阴影均为 3", "圆角与阴影均为 6", "圆角与阴影均为 10",
            "圆角与阴影均为 15", "圆角与阴影均为 20", "圆角与阴影均为 30", "圆角与阴影均为 50"};
    private int[] radiusArray = {3, 6, 10, 15, 20, 30, 50};
    class CardSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            int interval = radiusArray[arg2];
            cv_card.setRadius(interval);
            cv_card.setCardElevation(interval);
            MarginLayoutParams params = (MarginLayoutParams) cv_card.getLayoutParams();
            params.setMargins(interval, interval, interval, interval);
            cv_card.setLayoutParams(params);
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }
}
```

卡片视图的显示效果如图 13-7 和图 13-8 所示。其中，图 13-7 所示为阴影宽度为 6 的画面，此时卡片看起来比较薄；图 13-8 所示为阴影宽度为 15 的画面，此时卡片看起来比较厚。



图 13-7　阴影厚度为 6 的卡片视图



图 13-8　阴影厚度为 15 的卡片视图

介绍完卡片视图，再说明 Android5.0 引入的另一个新控件——调色板 Palette。调色板把多种颜色混合在一起，调和均匀后显示出新颜色。在 App 使用场景中常常会用到背景色调，即根据前景图片的总体色彩设置与之接近的背景色调，这样显得整个画面风格比较统一。例如，



对于喜庆的节日相片可设置偏红色调的背景，对于泛黄的老照片可设置偏黄色调的背景，对于山水风景的图片可设置偏绿色调的背景。根据每幅图片的色彩情况自动计算该图片的总体色调，通过调色板控件 Palette 就能完成。

因为 Palette 是 5.0 之后增加的新控件，所以要修改 build.gradle，在 dependencies 节点中加入下面一行代码表示导入 palette 库：

```
compile 'com.android.support:palette-v7:25.0.0'
```

下面是 Palette 的常用方法说明。

- from：从位图对象中获得调色板的构建对象。
- Builder.generate：给构建对象注册调色板的调色监听器，因为 Android 认为计算色调是耗时操作，得另外开线程处理，所以要注册监听器实现回调操作。调色监听器需实现接口 PaletteAsyncListener 的 onGenerated 方法，该方法在色调计算完毕后触发。
- getVibrantSwatch：获取偏亮色调的色板对象。调用色板对象的 getRgb 方法可得到具体颜色。
- getSwatches：获取所有色板对象。因为 getVibrantSwatch 方法有时会返回 null，此时要调用 getSwatches 方法取第一条颜色。

调色板的具体应用可跟卡片视图联合使用，也就是把调色板计算得到的色调填入卡片视图的边框背景中，从而实现卡片原图与边框的色彩呼应效果。

使用调色板的代码如下：

```
private void initPalette() {
    for (int i=0; i<mImageRes.length; i++) {
        Drawable drawable = getResources().getDrawable(mImageRes[i]);
        Bitmap bitmap = ((BitmapDrawable)drawable).getBitmap();
        Palette.Builder builder = Palette.from(bitmap);
        builder.generate(new MyPaletteListener(i));
    }
}

private class MyPaletteListener implements PaletteAsyncListener {
    private int mPos;
    public MyPaletteListener(int pos) {
        mPos = pos;
    }

    @Override
    public void onGenerated(Palette palette) {
        Palette.Swatch swatch = palette.getVibrantSwatch();
        if (swatch != null) {
            mBackColors[mPos] = swatch.getRgb();
        } else {    //getVibrantSwatch 有时会返回 null，此时从 getSwatches 取第一条颜色
            List<Palette.Swatch> swatches = palette.getSwatches();
```

```
                    for (int i=0; i<swatches.size(); i++) {
                        Palette.Swatch item = swatches.get(i);
                        mBackColors[mPos] = item.getRgb();
                        break;
                    }
                }
                gl_album.setAdapter(new AlbumAdapter(AlbumActivity.this, mImageRes, mBackColors));
            }
        }
```

学会了卡片视图与调色板的用法，剩下的工作便是精确加工相册的每张缩略图，给它们加上卡片边框，并给边框背景设置该缩略图的调和色。赶紧动手进行实践，体验一下自己的劳动成果带来的喜悦吧！

装饰后的相册效果如图 13-9 和图 13-10 所示。其中，图 13-9 所示为打开第一张图片时的相册画面，图 13-10 所示为打开最后一张图片的相册画面。



图 13-9　青青相册查看第一张图片



图 13-10　青青相册查看最后一张图片

至此，一个初具面貌的相册基本完工了。叫好的 App 还得有个好听的名称，笔者姑且将它命名为“青青相册”，读者看看要不要加上什么新功能，再取一个更好听的名字？

13.2　视频播放

本节介绍视频播放的相关技术，首先说明视频视图的工作原理，结合拖动条实现简单的视频播放器；接着阐述媒体控制条的用法，以及媒体控制条与视频视图的两种绑定方式；最后演示阶段性实战项目“影视播放器——爱看剧场”的改造方案和具体的实施细节。

13.2.1 视频视图 VideoView

第 9 章在介绍录像放映功能时使用了 MediaPlayer 结合 SurfaceView 播放视频文件，其中通过 SurfaceView 显示视频的画面，通过 MediaPlayer 设置播放参数并控制视频的播放操作。不过仅仅播一个视频就得如此深入掌握技术细节未免太兴师动众了，因此 Android 推出了视频视图 VideoView，该控件内部集成了 SurfaceView 和 MediaPlayer，从而实现视频画面与视频操作的统一管理，为开发者进行视频开发提供便利。

下面是 VideoView 的常用方法说明。

- setVideoPath：设置视频文件的路径。
- setMediaController：设置媒体控制条的对象。
- setOnPreparedListener：设置预备播放监听器。需实现监听器 OnPreparedListener 的 onPrepared 方法，该方法在准备播放时调用。
- setOnCompletionListener：设置结束播放监听器。需实现监听器 OnCompletionListener 的 onCompletion 方法，该方法在结束播放时调用。
- setOnErrorListener：设置播放异常监听器。需实现监听器 OnErrorListener 的 onError 方法，该方法在播放出现异常时调用。
- setOnInfoListener：设置播放信息监听器。需实现监听器 OnInfoListener 的 onInfo 方法，该方法在播放需要传递某种消息时调用，如开始/结束缓冲。
- requestFocus：请求获得焦点。该方法要在 start 方法前调用。
- start：开始播放视频。
- pause：暂停播放视频。
- resume：恢复播放视频。
- suspend：结束播放并释放资源。
- seekTo：拖动视频到指定进度开始播放。
- getDuration：获得视频的总时长。
- getCurrentPosition：获得当前的播放位置。该方法返回值与 getDuration 相等时，表示播放到了末尾。
- isPlaying：判断是否正在播放。
- getBufferPercentage：获得已缓冲的比例。返回值在 0 到 1 之间。

由于 VideoView 只是一个播放界面，本身不会显示进度条，因此实际开发中至少得给它配备一个拖动条 SeekBar，一方面用来展示当前的播放进度，另一方面用来拖动播放位置。

在 VideoView 的方法中，SeekBar 主要用到了三个方法，第一个 getDuration 方法获得的总时长对应拖动条的最大进度值，第二个 getCurrentPosition 方法对应拖动条的当前进度值，第三个 seekTo 方法是在用户拖动 SeekBar 结束后调用。为 VideoView 加上 SeekBar，即可实现基本的播放控制操作。

下面是使用 VideoView 结合 SeekBar 的代码，相比第 9 章的放映代码明显精简许多：

```
public class VideoViewActivity extends AppCompatActivity implements
        OnClickListener, FileSelectCallbacks, OnSeekBarChangeListener {
    private VideoView vv_play;
    private SeekBar sb_play;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_video_view);
        findViewById(R.id.btn_open).setOnClickListener(this);
        vv_play = (VideoView) findViewById(R.id.vv_play);
        sb_play = (SeekBar) findViewById(R.id.sb_play);
        sb_play.setOnSeekBarChangeListener(this);
        sb_play.setEnabled(false);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_open) {
            FileSelectFragment.show(this, new String[] { "mp4" }, null);
        }
    }

    @Override
    public void onConfirmSelect(String absolutePath, String fileName, Map<String, Object> map_param) {
        String file_path = absolutePath + "/" + fileName;
        vv_play.setVideoPath(file_path);
        vv_play.requestFocus();
        vv_play.start();
        sb_play.setEnabled(true);
        mHandler.post(mRefresh);
    }

    @Override
    public boolean isFileValid(String absolutePath, String fileName, Map<String, Object> map_param) {
        return true;
    }

    private Handler mHandler = new Handler();
    private Runnable mRefresh = new Runnable() {
        @Override
        public void run() {
            sb_play.setProgress(100 * vv_play.getCurrentPosition()/vv_play.getDuration());
            mHandler.postDelayed(this, 500);
        }
```

```
        };

        @Override
        public void onProgressChanged(SeekBar seekBar, int progress,boolean fromUser) {
        }

        @Override
        public void onStartTrackingTouch(SeekBar seekBar) {
        }

        @Override
        public void onStopTrackingTouch(SeekBar seekBar) {
            int pos = seekBar.getProgress() * vv_play.getDuration() / 100;
            vv_play.seekTo(pos);
        }
    }
```

VideoView 与 SeekBar 的播放效果如图 13-11 和图 13-12 所示。其中，图 13-11 所示为视频播放开始的画面，此时拖动条的进度图标尚在左边；图 13-12 所示为视频播放即将结束的画面，此时拖动条的进度图标已经移到右边了。



图 13-11　视频视图刚刚开始播放



图 13-12　视频视图即将结束播放

13.2.2　媒体控制条 MediaController

使用拖动条主要完成两个播放控制功能：显示当前播放进度和拖动到指定位置播放。这两个基本功能显然不够全面，对于一个视频播放器来说，至少还得实现下列基础功能：

（1）暂停功能和暂停之后的恢复播放功能。
（2）查看视频的总时长和当前已播放的时长。
（3）快进和快退功能。

前面介绍 VideoView 的常用方法时提到 setMediaController 方法可设置媒体控制条的对象，这个媒体控制条就是 MediaController，它的界面跟 Windows 上的播放条几乎一模一样，并支持一些基本的播放控制操作：显示当前的播放进度、拖动到指定位置播放、暂停播放与恢复播放、查看视频的总时长和已播放时长、对视频做快进或快退操作等。



下面是 MediaController 的常用方法说明。

- setMediaPlayer：设置媒体播放器的对象，即 VideoView 对象。该方法与 setAnchorView 只能调用一个。
- setAnchorView：设置绑定的主视图，其实一般是一个 VideoView 对象。该方法与 setMediaPlayer 只能调用一个。
- show：显示媒体控制条。
- hide：隐藏媒体控制条。
- isShowing：判断媒体控制条是否正在显示。
- setPrevNextListeners：设置前一个按钮与后一个按钮的点击监听器（OnClickListener）。如果没调用该方法，那么前一个按钮与后一个按钮都不会展示。

VideoView 继承自 SurfaceView，而 MediaController 继承自 FrameLayout，理论上这两个控件是可以随意摆放的，但是考虑到用户的使用习惯，往往将其集成在一起展示，即媒体控制条固定放在视频视图的底部。因此无须在布局文件中声明 MediaController 控件，只需声明 VideoView 控件，然后在代码中将媒体控制条附着于视频视图即可。甚至布局文件中都不用声明 VideoView，在代码中动态添加视频视图和媒体控制条即可。由此衍生出 VideoView 与 MediaController 的两种集成方式：在布局文件中声明 VideoView 和在代码中动态添加 VideoView。

1. 在布局文件中声明 VideoView

视频视图对象的使用步骤不变，即先调用 setVideoPath 方法指定视频文件，然后调用 setMediaController 方法指定控制条，最后调用 start 方法开始播放。此时媒体控制条对象在完成构建后只需调用 setMediaPlayer 方法设置播放器对象即可。

该方式的控件集成代码如下：

```
public void onConfirmSelect(String absolutePath, String fileName, Map<String, Object> map_param) {
    String file_path = absolutePath + "/" + fileName;
    vv_play.setVideoPath(file_path);
    vv_play.requestFocus();
    //媒体控制条代码开始
    MediaController mc_play = new MediaController(this);
    vv_play.setMediaController(mc_play);
    mc_play.setMediaPlayer(vv_play);
    //媒体控制条代码结束
    vv_play.start();
}
```

2. 在代码中动态添加 VideoView

视频视图对象需要在代码中构建并添加，其余使用步骤同上。此时媒体控制条对象的使用步骤发生变化，不再调用 setMediaPlayer 方法，而改成调用 setAnchorView 方法，该方法把



媒体控制条添加到宿主视图上，如果方法参数是一个 VideoView 对象，就将媒体控制条添加到 VideoView 的上级视图。

该方式的控件集成代码如下：

```
public void onConfirmSelect(String absolutePath, String fileName, Map<String, Object> map_param) {
        String file_path = absolutePath + "/" + fileName;
        VideoView vv_play = new VideoView(this);
        vv_play.setVideoPath(file_path);
        vv_play.requestFocus();
        MediaController mc_play = new MediaController(this);
        mc_play.setAnchorView(vv_play);
        mc_play.setKeepScreenOn(true);
        vv_play.setMediaController(mc_play);
        ll_play.addView(vv_play);
        vv_play.start();
}
```

两种集成方式的屏幕画面基本一致，开发者可根据视频的展示位置决定采用哪一种方式。视频播放开始时不会显示媒体控制条，只有用户点击视频画面后才会弹出控制条；如果过了几秒没有操作控制条，它就会自动消失。

集成后的播放效果如图 13-13 和图 13-14 所示。其中，图 13-13 所示为视频播放一开始的画面，不点击视频画面就不会出现媒体控制条；图 13-14 所示为点击视频画面后的截图，点击后弹出媒体控制条，即可进行视频播放的控制操作。



图 13-13　媒体控制条已隐藏



图 13-14　媒体控制条已弹出来

13.2.3　影视播放器——爱看剧场

从前面 VideoView 和 MediaController 的集成效果来看，这个视频播放器只提供基本的播放控制，欠缺许多高级播放功效，包括：

（1）控制条分上下两行，上面是控制按钮，下面是进度条，高度太宽，有碍观瞻。

（2）按钮样式无法定制，不能增加新按钮，也无法删除按钮。

（3）进度条与播放时间的样式也不能定制。
（4）播放器的视频画面不会自动全屏显示。
（5）播放器无法控制调大和调小音量。
（6）播放器不会自动设置标题和背景。

基于以上缺陷，要想让视频播放画面生动活泼起来，势必要重新写一个既好看又好用的播放器。这里既要对视频视图 VideoView 进行重写，又要对媒体控制条 MediaController 进行重写。经过进一步查看源码与深入分析，我们发现播放器的改造内容主要分为两个方面，一方面是对视频画面做功能方面的增强，另一方面是对控制条做样式方面的定制。视频视图和媒体播放条的改造方案基本确定如下：

1. 增强 VideoView 的功能

可以派生一个电影视图 MovieView，并提供以下新增功能：

（1）重写尺寸测量方法 onMeasure，实现自动全屏。
（2）重写触摸事件监听方法 onTouch，用于弹出或关闭视频控制条。
（3）重写按键事件监听方法 onKeyDown，用于调节音量的大小。
（4）补充新方法用于设置标题和背景。

2. 定制 MediaController 的样式

由于媒体控制条的内部控件都是私有的，即使继承了也无法修改，因此只能自己写一个全新的视频控制条 VideoController。好在需求只是更改控制条的样式，并未增加复杂功能，所以视频控制条提供以下控件即可：

（1）一个播放按钮，点击按钮暂停播放，再次点击恢复播放。
（2）一个拖动条，动态显示当前播放进度，允许把视频拖动到指定位置开始播放。
（3）两个文本控件，一个显示视频的总时长，另一个显示视频的已播放时长。

视频控制条的难点在于如何跟电影视图同步当前的播放进度。对于电影视图向控制条通知播放进度，可以设置定时器持续刷新播放进度；对于控制条向电影视图通知播放动作，可以监听播放按钮的点击事件和拖动条的拖动事件，并将事件处理结果传给电影视图 MovieView 对象。

如果只修改代码，还不能完全实现自动全屏功能，主要问题如下：

（1）手机屏幕顶部的系统状态栏依然停留在屏幕顶端。
（2）App 自身的导航栏没有隐藏。
（3）在视频播放途中，如果手机屏幕发生切换操作（如从竖屏变为横屏），视频播放就会停止，回到播放页面刚进去的初始画面。

对于前两个问题，可增加一个全屏的页面风格，并给视频播放页面的 activity 节点设置 android:theme 属性予以调整。全屏风格的内容包括设置属性 android:windowFullscreen 隐藏系统状态栏、设置属性 android:windowNoTitle 去除 App 的导航栏、设置属性 android:

windowContentOverlay 清除窗体背景。全屏风格的具体设置如下：

```
<style name="FullScreenTheme" parent="AppBaseTheme">
    <item name="android:windowFullscreen">true</item>
    <item name="android:windowNoTitle">true</item>
    <item name="android:windowContentOverlay">@null</item>
</style>
```

对于第 3 个问题，可给 activity 节点增加属性 android:configChanges，并设置合适的属性值加以预防。因为默认情况下，App 每次切换屏幕都会重启 Activity，即先执行活动页面的 onDestroy 方法，再执行活动页面的 onCreate 方法（具体参见第 3 章的 Activity 生命周期），这便导致正在播放的视频被中断返回了。新增属性 configChanges 的意思是，在某些情况下，屏幕变更不用重启活动页面，只需调用 onConfigurationChanged 方法重新设定显示方式。所以只要给该属性指定若干豁免情况，就能避免无谓的页面重启操作。屏幕变更豁免情况的取值说明见表 13-2。

表13-2 屏幕变更豁免情况的取值说明

configChanges 属性的取值	说明
touchscreen	触摸屏发生改变，一般不会发生
keyboard	键盘发生改变，例如使用了外部键盘
keyboardHidden	软键盘弹出或隐藏
navigation	导航发生改变，一般不会发生
screenLayout	屏幕的显示发生改变，例如使用了外部显示器
fontScale	字体比例发生改变，例如在系统设置中调整默认字体
orientation	屏幕发生横屏与竖屏的切换
screenSize	屏幕大小发生改变

另外，要新增 screenOrientation 属性，表明该页面允许哪种形式的屏幕方向。对于视频播放页面来说，该属性值要设置为 sensor，表示由传感器控制。屏幕方向的取值说明见表 13-3。

表13-3 屏幕方向的取值说明

screenOrientation 属性的取值	说明
portrait	只允许垂直方向
landscape	只允许水平方向
sensor	由传感器控制方向
unspecified	默认值，由系统选择方向
user	使用用户当前首选的方向
fullSensor	显示的 4 个方向由传感器决定，即 4 个方向都允许倒转

下面是视频播放页面的 activity 节点配置：

```
<activity
    android:name=".MoviePlayerActivity"
```

```
            android:configChanges="orientation|keyboardHidden|screenSize"
            android:screenOrientation="sensor"
            android:theme="@style/FullScreenTheme" >
        </activity>
```

改造后的电影视图 MovieView 代码如下：

```
//支持以下功能 ：自动全屏、调节音量、收缩控制栏、设置背景
@TargetApi(Build.VERSION_CODES.JELLY_BEAN)   //setBackground 需要
public class MovieView extends VideoView implements OnTouchListener,OnKeyListener,VolumeAdjustListener {
    private static final String TAG = "MovieView";
    private Context mContext;
    private int screenWidth, screenHeight;
    private int videoWidth, videoHeight;
    private int realWidth, realHeight;
    private int mXpos, mYpos, mOffset;
    // 自动隐藏顶部和底部 View 的时间
    public static final int HIDE_TIME = 5000;
    private View mTopView;
    private View mBottomView;
    private AudioManager mAudioMgr;
    private VolumeDialog dialog;
    private Handler mHandler = new Handler();

    public MovieView(Context context) {
        this(context, null);
    }

    public MovieView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public MovieView(Context context, AttributeSet attrs, int defStyle) {
        super(context, attrs, defStyle);
        mContext = context;
        screenWidth = DisplayUtil.getSreenWidth(mContext);
        screenHeight = DisplayUtil.getSreenHeight(mContext);
        mOffset = Utils.dip2px(mContext, 10);
        mAudioMgr = (AudioManager) mContext.getSystemService(Context.AUDIO_SERVICE);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
        int width = getDefaultSize(realWidth, widthMeasureSpec);
```

```
        int height = getDefaultSize(realHeight, heightMeasureSpec);
        if (realWidth > 0 && realHeight > 0) {
            if (realWidth * height > width * realHeight) {
                height = width * realHeight / realWidth;
            } else if (realWidth * height < width * realHeight) {
                width = height * realWidth / realHeight;
            }
        }
        setMeasuredDimension(width, height);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            mXpos = (int) event.getX();
            mYpos = (int) event.getY();
            break;
        case MotionEvent.ACTION_UP:
            if (Math.abs(event.getX()-mXpos) < mOffset && Math.abs(event.getY()-mYpos) < mOffset) {
                showOrHide();
            }
            break;
        default:
            break;
        }
        return true;
    }

    public void prepare(View topTiew, View bottomView) {
        mTopView = topTiew;
        mBottomView = bottomView;
        setBackgroundResource(R.drawable.video_bg1);
    }

    public void begin(MediaPlayer mp) {
        setBackground(null);
        if (mp != null) {
            videoWidth = mp.getVideoWidth();
            videoHeight = mp.getVideoHeight();
        }
        realWidth = videoWidth;
        realHeight = videoHeight;
```

```
            start();
    }

    public void end(MediaPlayer mp) {
        setBackgroundResource(R.drawable.video_bg3);
        realWidth = screenWidth;
        realHeight = screenHeight;
    }

    public void showOrHide() {
        if (mTopView==null || mBottomView==null) {
            return;
        }
        if (mTopView.getVisibility() == View.VISIBLE) {
            mTopView.clearAnimation();
            Animation animTop = AnimationUtils.loadAnimation(mContext, R.anim.leave_from_top);
            animTop.setAnimationListener(new AnimationImp() {
                @Override
                public void onAnimationEnd(Animation animation) {
                    mTopView.setVisibility(View.GONE);
                }
            });
            mTopView.startAnimation(animTop);
            mBottomView.clearAnimation();
            Animation animBottom = AnimationUtils.loadAnimation(mContext, R.anim.leave_from_bottom);
            animBottom.setAnimationListener(new AnimationImp() {
                @Override
                public void onAnimationEnd(Animation animation) {
                    mBottomView.setVisibility(View.GONE);
                }
            });
            mBottomView.startAnimation(animBottom);
        } else {
            mTopView.setVisibility(View.VISIBLE);
            mTopView.clearAnimation();
            Animation animTop = AnimationUtils.loadAnimation(mContext, R.anim.entry_from_top);
            mTopView.startAnimation(animTop);
            mBottomView.setVisibility(View.VISIBLE);
            mBottomView.clearAnimation();
            Animation animBottom = AnimationUtils.loadAnimation(mContext, R.anim.entry_from_bottom);
            mBottomView.startAnimation(animBottom);
            mHandler.removeCallbacks(mHide);
            mHandler.postDelayed(mHide, HIDE_TIME);
```



```
        }
    }

    private Runnable mHide = new Runnable() {
        @Override
        public void run() {
            showOrHide();
        }
    };

    private class AnimationImp implements AnimationListener {
        @Override
        public void onAnimationEnd(Animation animation) {
        }

        @Override
        public void onAnimationRepeat(Animation animation) {
        }

        @Override
        public void onAnimationStart(Animation animation) {
        }
    }

    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_VOLUME_UP) {
            showVolumeDialog(AudioManager.ADJUST_RAISE);
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {
            showVolumeDialog(AudioManager.ADJUST_LOWER);
            return true;
        }
        return false;
    }

    private void showVolumeDialog(int direction) {
        if (dialog==null || dialog.isShowing()!=true) {
            dialog = new VolumeDialog(mContext);
            dialog.setVolumeAdjustListener(this);
            dialog.show();
        }
        dialog.adjustVolume(direction, true);
```

```
            onVolumeAdjust(mAudioMgr.getStreamVolume(AudioManager.STREAM_MUSIC));
        }

        @Override
        public void onVolumeAdjust(int volume) {
        }
    }
```

最终的影视播放器效果如图 13-15～图 13-18 所示。其中，图 13-15 所示为播放器启动的初始画面，一开始没有打开视频，先显示默认背景；图 13-16 所示为打开视频开始播放的画面，此时视频上部展示标题栏、下部展示控制条。



图 13-15　爱看剧场初始画面

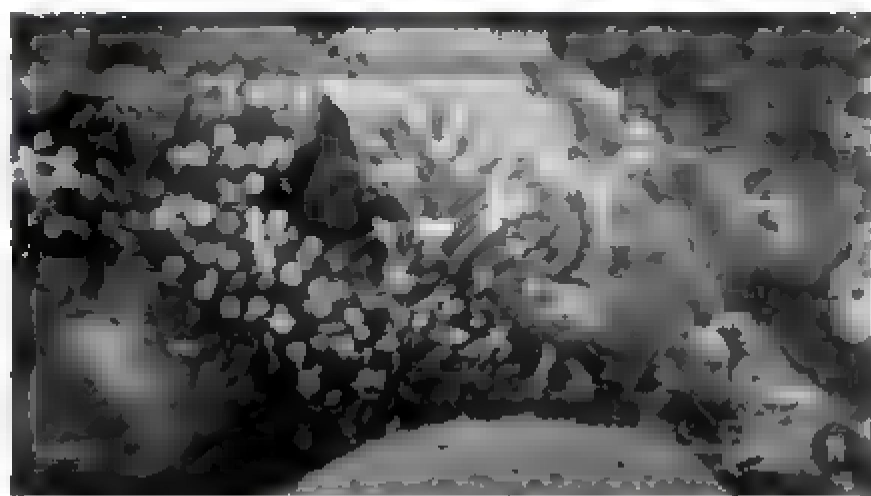


图 13-16　爱看剧场开始播放

如果播放过程中不做任何操作，标题栏与控制条等待五秒后就会自动隐藏，如图 13-17 所示。如果需要调节音量大小，就按手机侧面的加、减按钮，屏幕中央会自动弹出音量对话框，如图 13-18 所示。音量对话框的实现过程参见第 11 章。

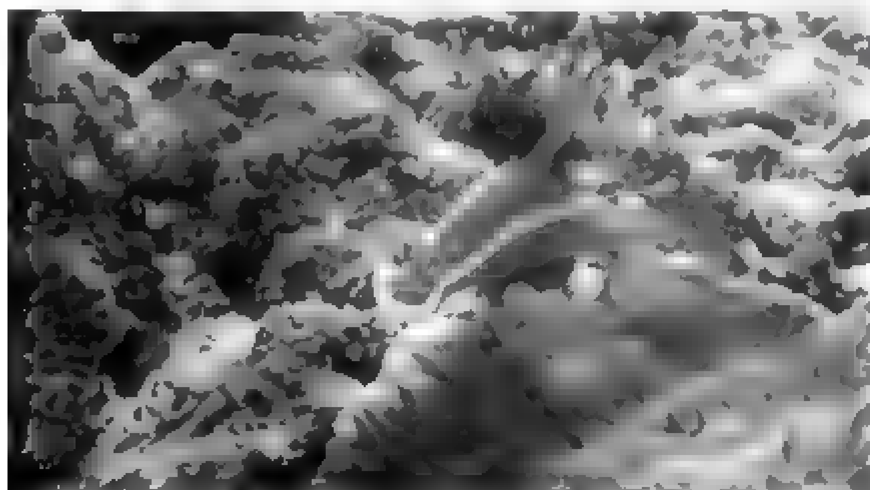


图 13-17　爱看剧场正在播放

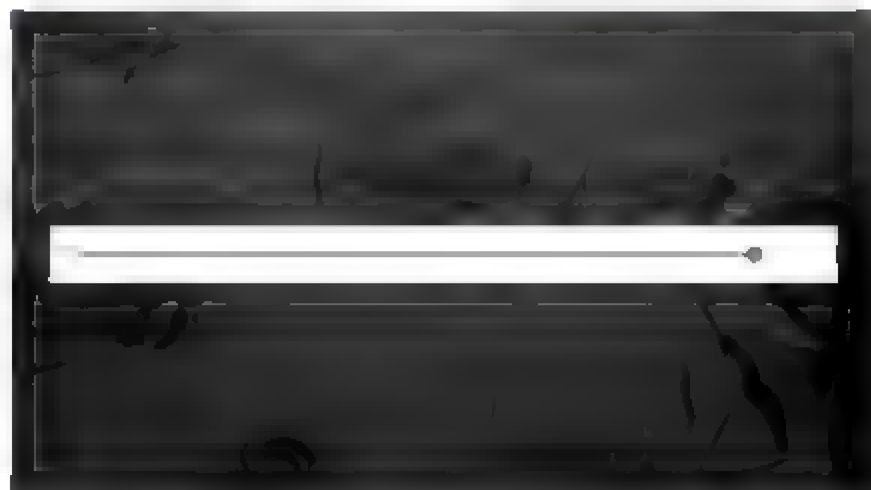


图 13-18　爱看剧场调节音量

经过一番改造，这个影视播放器总算满足了大部分日常播放要求，这也是主流视频播放器必备的播放功能。怎么样，是不是颇有成就感呢？该播放器作为阶段性的实战项目，给取一个大名，叫“爱看剧场”好了。

13.3　内容提供与处理

本节介绍 Android 四大组件之一的 ContentProvider 的基本概念和常见用法。首先说明如何使用内容提供器封装数据的外部访问接口；接着阐述如何通过内容解析器在外部查询和修改数

据，以及使用内容操作器完成批量数据操作；然后说明内容观察器的应用场合，并演示如何借助内容观察器实现流量校准的功能。

13.3.1 内容提供器 ContentProvider

Android 号称提供了四大组件，分别是页面 Activity、广播 Broadcast、服务 Service 和内容提供器 ContentProvider，前 3 个组件已经分别在第 3 章、第 5 章、第 6 章做了详细介绍，现在总算轮到 ContentProvider 了，之所以这么迟提到内容提供器，是因为它的使用场合相对有限，往往容易被人直接忽略。

ContentProvider 为 App 存取内部数据提供统一的外部接口，让不同的应用之间得以共享数据。像我们熟知的 SQLite 操作的是应用自身的内部数据库；文件的上传和下载操作的是后端服务器的文件；而 ContentProvider 操作的是本设备其他应用的内部数据，是一种中间层次的数据存储形式。

在实际编码中，ContentProvider 只是一个服务端的数据存取接口，开发者需要在其基础上实现一个具体类，并重写以下相关数据库管理方法。

- onCreate：创建数据库并获得数据库连接。
- query：查询数据。
- insert：插入数据。
- update：更新数据。
- delete：删除数据。
- getType：获取数据类型。

这些方法看起来是不是很像 SQLite？没错，ContentProvider 作为中间接口，本身并不直接保存数据，而是通过 SQLiteOpenHelper 与 SQLiteDatabase 间接操作底层的 SQLite。所以要想使用 ContentProvider，首先得实现 SQLite 的数据表帮助类，然后由 ContentProvider 封装对外的接口。

下面是使用 ContentProvider 提供用户信息对外接口的代码：

```
public class UserInfoProvider extends ContentProvider {
    public static final int USER_INFO = 1;
    public static final UriMatcher uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    // 这里的 AUTHORITIES 取值必须与 AndroidManifest.xml 里的 android:authorities 保持一致
    static {
        uriMatcher.addURI(UserInfoContent.AUTHORITIES, "/user", USER_INFO);
    }
    private UserDBHelper userDB;

    //删除数据
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        int count = 0;
```

```
        if (uriMatcher.match(uri) == USER_INFO) {
            SQLiteDatabase db = userDB.getWritableDatabase();
            count = db.delete(UserInfoContent.TABLE_NAME, selection, selectionArgs);
            db.close();
        }
        return count;
    }

    //插入数据
    @Override
    public Uri insert(Uri uri, ContentValues values) {
        Uri newUri = uri;
        if (uriMatcher.match(uri) == USER_INFO) {
            SQLiteDatabase db = userDB.getWritableDatabase();
            // 向指定的表插入数据，得到返回的 id
            long rowId = db.insert(UserInfoContent.TABLE_NAME, null, values);
            if (rowId > 0) {// 判断插入是否执行成功
                // 如果添加成功，就利用新添加的 id 和新生成的地址
                newUri = ContentUris.withAppendedId( UserInfoContent.CONTENT_URI, rowId);
                // 通知监听器，数据已经改变
                getContext().getContentResolver().notifyChange(newUri, null);
            }
            db.close();
        }
        return uri;
    }

    //创建 ContentProvider 时调用的回调函数
    @Override
    public boolean onCreate() {
        userDB = UserDBHelper.getInstance(getContext(), 1);
        return false;
    }

    //查询数据库
    @Override
    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
        Cursor cursor = null;
        if (uriMatcher.match(uri) == USER_INFO) {
            SQLiteDatabase db = userDB.getReadableDatabase();
            // 执行查询
            cursor = db.query(UserInfoContent.TABLE_NAME,
                    projection, selection, selectionArgs, null, null, sortOrder);
```



```
            // 设置监听
            cursor.setNotificationUri(getContext().getContentResolver(), uri);
        }
        return cursor;
    }

    //数据访问类型，暂未实现
    @Override
    public String getType(Uri uri) {
        throw new UnsupportedOperationException("Not yet implemented");
    }

    //更新数据，暂未实现
    @Override
    public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs) {
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

既然内容提供器是四大组件之一，就得在 AndroidManifest.xml 中注册它的定义，并开放外部访问权限，注册代码如下：

```
<provider
    android:name=".provider.UserInfoProvider"
    android:authorities="com.example.media.provider.UserInfoProvider"
    android:enabled="true"
    android:exported="true" />
```

注册完毕后就完成了服务端 App 的封装工作，接下来可由其他 App 进行数据存取。

13.3.2 内容解析器 ContentResolver

前面提到了利用 ContentProvider 实现服务端 App 的数据封装，如果客户端 App 想访问对方的内部数据，就要通过内容解析器 ContentResolver 访问。内容解析器是客户端 App 操作服务端数据的工具，相对应的内容提供器是服务端的数据接口。要获取 ContentResolver 对象，在 Activity 代码中调用 getContentResolver 方法即可。

ContentResolver 提供的方法与 ContentProvider 是一一对应的，比如 query、insert、update、delete、getType 等方法，连方法的参数类型都一模一样。其中，最常用的是 query 函数，调用该函数返回一个游标 Cursor 对象，这个游标与 SQLite 的游标是一样的，想必读者早已用得炉火纯青。

下面是 query 方法的具体参数说明（依顺序排列）。

- uri：Uri 类型，可以理解为本次操作的数据表路径。
- projection：字符串数组类型，指定将要查询的字段名称列表。

- selection：字符串类型，指定查询条件。
- selectionArgs：字符串数组类型，指定查询条件中的参数取值列表。
- sortOrder：字符串类型，指定排序条件。

针对前面 UserInfoProvider 提供的数据接口，下面使用 ContentResolver 在客户端添加用户信息，代码如下：

```
private void addUser(ContentResolver resolver, UserInfo user) {
    ContentValues name = new ContentValues();
    name.put("name", user.name);
    name.put("age", user.age);
    name.put("height", user.height);
    name.put("weight", user.weight);
    name.put("married", false);
    name.put("update_time", Utils.getNowDateTime());
    // insert 的第一个参数指明了要访问的数据源
    resolver.insert(UserInfoContent.CONTENT_URI, name);
}
```

下面是使用 ContentResolver 在客户端查询所有用户信息的代码：

```
private String readAllUser(ContentResolver resolver) {
    ArrayList<UserInfo> userArray = new ArrayList<UserInfo>();
    // query 的第一个参数指明了要访问的数据源
    Cursor cursor = resolver.query(UserInfoContent.CONTENT_URI, null, null, null, null);
    while (cursor.moveToNext()) {
        UserInfo user = new UserInfo();
        user.name = cursor.getString(cursor.getColumnIndex(UserInfoContent.USER_NAME));
        user.age = cursor.getInt(cursor.getColumnIndex(UserInfoContent.USER_AGE));
        user.height = cursor.getInt(cursor.getColumnIndex(UserInfoContent.USER_HEIGHT));
        user.weight = cursor.getFloat(cursor.getColumnIndex(UserInfoContent.USER_WEIGHT));
        userArray.add(user);
    }
    cursor.close();
    String result = "";
    for (int i=0; i<userArray.size(); i++) {
        UserInfo user = userArray.get(i);
        result = String.format("%s%s    年龄%d   身高%d   体重%f\n", result,
                user.name, user.age, user.height, user.weight);
    }
    return result;
}
```

添加用户信息的效果如图 13-19 所示，一开始服务端的用户表不存在用户记录，客户端使用 ContentResolver 添加一条记录后，服务端的用户记录数返回 1。用户信息的查询明细如图



13-20所示，点击页面上的用户记录数量文字，弹出一个对话框，提示当前找到的所有用户的明细数据，包括姓名、年龄、身高、体重等信息。

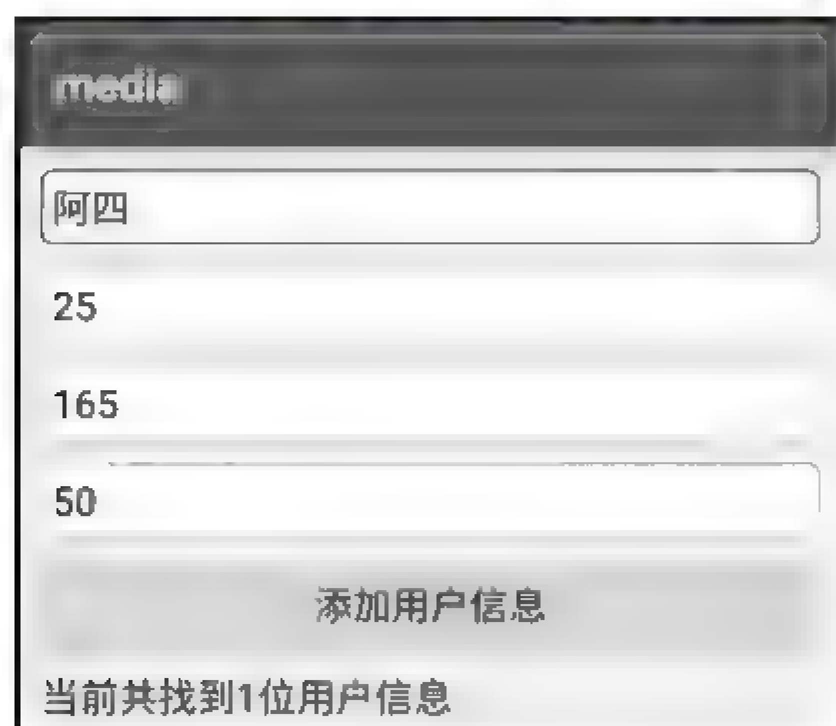


图 13-19 利用内容提供器添加用户信息

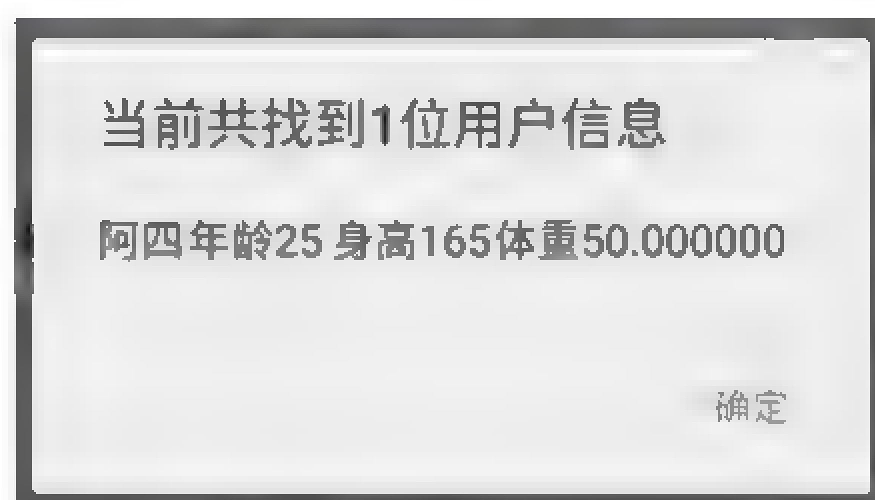


图 13-20 利用内容解析器查询获得用户信息

在实际开发中，普通 App 很少会开放数据接口给其他应用访问，作为服务端接口的 ContentProvider 基本用不到。内容组件能够派上用场的情况往往是 App 想要访问系统应用的通信数据，比如查看联系人、短信、通话记录，以及对这些通信信息进行增、删、改、查。

下面是使用 ContentResolver 添加联系人信息的代码片段，此时访问的数据来源变成了系统自带的 raw_contacts：

```
public static void addContacts(ContentResolver resolver, Contact contact) {
    // 往 raw_contacts 中添加数据，并获取添加的 id 号
    Uri raw_uri = Uri.parse("content://com.android.contacts/raw_contacts");
    ContentValues values = new ContentValues();
    long contactId = ContentUris.parseId(resolver.insert(raw_uri, values));

    // 往 data 中添加数据（要根据前面获取的 id 号）
    Uri uri = Uri.parse("content://com.android.contacts/data");
    ContentValues name = new ContentValues();
    name.put("raw_contact_id", contactId);
    name.put("mimetype", "vnd.android.cursor.item/name");
    name.put("data2", contact.name);
    resolver.insert(uri, name);

    ContentValues phone = new ContentValues();
    phone.put("raw_contact_id", contactId);
    phone.put("mimetype", "vnd.android.cursor.item/phone_v2");
    phone.put("data2", "2");
    phone.put("data1", contact.phone);
    resolver.insert(uri, phone);

    ContentValues email = new ContentValues();
    email.put("raw_contact_id", contactId);
    email.put("mimetype", "vnd.android.cursor.item/email_v2");
```

```
        email.put("data2", "2");
        email.put("data1", contact.email);
        resolver.insert(uri, email);
    }
```

注意上述代码用了 4 条 insert 语句，但业务上只添加了一个联系人信息。这样处理有一个问题，就是 4 个 insert 操作不在同一个事务中，要是中间某步 insert 操作失败，那么之前插入成功的记录就无法自动回滚，从而产生垃圾数据。

为了避免这种情况的发生，Android 提供了内容操作器 ContentProviderOperation 进行批量数据的处理，即在一个请求中封装多条记录的修改动作，然后一次性提交给服务端，从而实现在一个事务中完成多条数据的更新操作。即使某条记录处理失败，ContentProviderOperation 也能根据事务一致性原则自动回滚本事务已经执行的修改操作。

下面是使用 ContentProviderOperation 批量添加联系人信息的代码片段：

```
    public static void addFullContacts(ContentResolver resolver, Contact contact) {
        Uri raw_uri = Uri.parse("content://com.android.contacts/raw_contacts");
        Uri uri = Uri.parse("content://com.android.contacts/data");
        ContentProviderOperation op_main = ContentProviderOperation
                .newInsert(raw_uri).withValue("account_name", null).build();
        ContentProviderOperation op_name = ContentProviderOperation
                .newInsert(uri).withValueBackReference("raw_contact_id", 0)
                .withValue("mimetype", "vnd.android.cursor.item/name")
                .withValue("data2", contact.name).build();
        ContentProviderOperation op_phone = ContentProviderOperation
                .newInsert(uri).withValueBackReference("raw_contact_id", 0)
                .withValue("mimetype", "vnd.android.cursor.item/phone_v2")
                .withValue("data2", "2").withValue("data1", contact.phone).build();
        ContentProviderOperation op_email = ContentProviderOperation
                .newInsert(uri).withValueBackReference("raw_contact_id", 0)
                .withValue("mimetype", "vnd.android.cursor.item/email_v2")
                .withValue("data2", "2").withValue("data1", contact.email).build();
        ArrayList<ContentProviderOperation> operations = new ArrayList<ContentProviderOperation>();
        operations.add(op_main);
        operations.add(op_name);
        operations.add(op_phone);
        operations.add(op_email);
        try {
            resolver.applyBatch("com.android.contacts", operations);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

添加联系人信息的效果如图 13-21 和图 13-22 所示。其中，图 13-21 所示为添加之前的截图，此时联系人个数为 157 位；图 13-22 所示为添加成功之后的截图，此时联系人个数为 158 位。



图 13-21 联系人添加之前的界面



图 13-22 联系人添加之后的界面

13.3.3 内容观察器 ContentObserver

ContentResolver 获取数据采用的是主动查询方式，有查询就有数据，没查询就没数据。有时我们不但要获取以往的数据，还要实时获取新增的数据，最常见的业务场景是短信验证码。电商 App 经常在用户注册或付款时发送验证码短信，为了给用户省事，App 通常会监控手机刚收到的验证码数字，并自动填入验证码输入框。这时就用到了内容观察器 ContentObserver，给目标内容注册一个观察器，目标内容的数据一旦发生变化，观察器规定好的动作马上触发，从而执行开发者预先定义的代码。

内容观察器的用法与内容提供器类似，也要从 ContentObserver 派生一个观察器类，然后通过 ContentResolver 对象调用相应的方法注册或注销观察器。下面是 ContentResolver 与观察器有关的方法说明。

- registerContentObserver：注册内容观察器。
- unregisterContentObserver：注销内容观察器。
- notifyChange：通知内容观察器发生了数据变化。

为了让读者更好理解，下面举一个实际应用的例子。在第 6 章的实战项目“手机安全助手”中，每月的流量限额由用户手动配置，但流量限额其实是由移动运营商指定的。以中国移动为例，只要发送流量校准短信给运营商客服号码（如发送 18 到 10086），运营商就会给用户发送本月的流量数据，包括月流量额度、已使用流量、未使用流量等信息。安全助手只需监控 10086 发送的短信内容，即可自动获取手机号码的月流量额度，无须用户手工配置。

下面是利用 ContentObserver 实现流量校准的代码片段：

```
private Handler mHandler = new Handler();
private SmsGetObserver mObserver;
private static Uri mSmsUri;
private static String[] mSmsColumn;

@TargetApi(Build.VERSION_CODES.KITKAT)
```

```
private void initSmsObserver() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
        mSmsUri = Telephony.Sms.Inbox.CONTENT_URI;
        mSmsColumn = new String[] {
                Telephony.Sms.ADDRESS, Telephony.Sms.BODY, Telephony.Sms.DATE };
    } else {
        mSmsUri = Uri.parse("content://sms/inbox");
        mSmsColumn = new String[] { "address", "body", "date" };
    }
    mObserver = new SmsGetObserver(this, mHandler);
    getContentResolver().registerContentObserver(mSmsUri, true, mObserver);
}

@Override
protected void onDestroy() {
    getContentResolver().unregisterContentObserver(mObserver);
    super.onDestroy();
};

private static class SmsGetObserver extends ContentObserver {
    private Context mContext;
    public SmsGetObserver(Context context, Handler handler) {
        super(handler);
        mContext = context;
    }

    @Override
    public void onChange(boolean selfChange) {
        String sender = "";
        String content = "";
        String selection = String.format("address='10086' and date>%d",
                System.currentTimeMillis()-1000*60*60);
        Cursor cursor = mContext.getContentResolver().query(
                mSmsUri, mSmsColumn, selection, null, " date desc");
        while(cursor.moveToNext()){
            sender = cursor.getString(0);
            content = cursor.getString(1);
            break;
        }
        cursor.close();
        if (pd != null && pd.isShowing() == true) {
            pd.dismiss();
        }
```

```
                //回调短信监听方法
                mCheckResult = String.format("发送号码 ：%s\n 短信内容 ：%s", sender, content);
                Log.d(TAG, "result="+mCheckResult);
                String flow = String.format("流量校准结果如下 ：\n\t 总流量为 ：%s\n\t 已使用 ：%s\n\t
剩余 ：%s",
                        findFlow(content, "总流量为", "MB"),
                        findFlow(content, "已使用", "MB"), findFlow(content, "剩余", "MB"));
                tv_check_flow.setText(flow);
                super.onChange(selfChange);
            }
        }

        private static String findFlow(String sms, String begin, String end) {
            int begin_pos = sms.indexOf(begin);
            if (begin_pos < 0) {
                return "未获取";
            }
            String sub_sms = sms.substring(begin_pos);
            int end_pos = sub_sms.indexOf(end);
            if (end_pos < 0) {
                return "未获取";
            }
            String flow_desc = sub_sms.substring(begin.length(), end_pos+end.length());
            return flow_desc;
        }
```

流量校准的效果如图 13-23 和图 13-24 所示。其中，图 13-23 所示为用户实际收到的短信内容，图 13-24 所示为 App 监视短信并解析完成的流量数据页面。

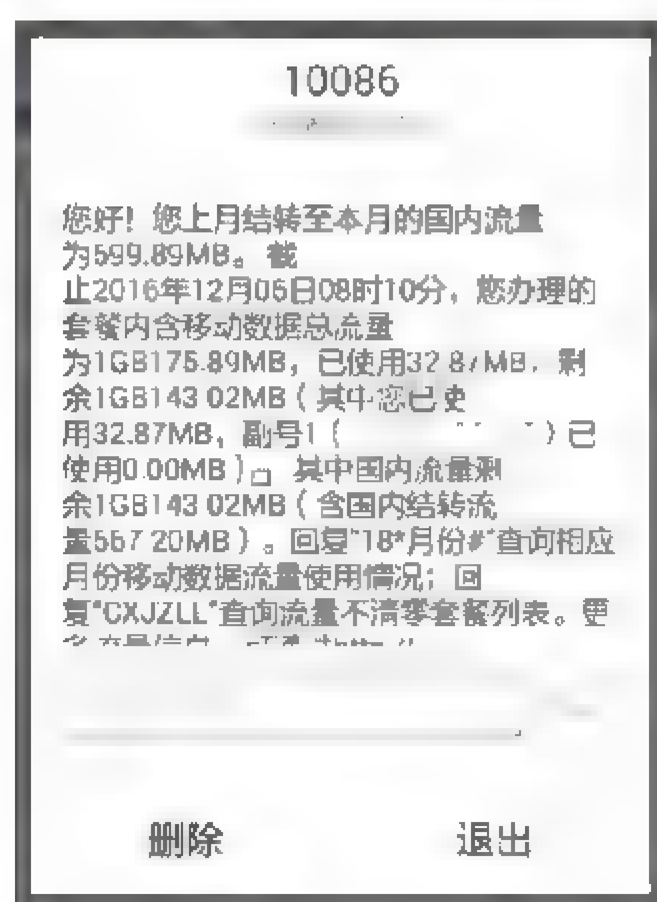


图 13-23　用户收到的短信内容



图 13-24　内容观察器监视并解析得到的流量信息

总结一下在 Content 组件经常使用的系统 URI，详细的 URI 取值说明见表 13-4。

表13-4 常用的系统URI取值说明

内容名称	URI 常量名	实际路径
联系人	ContactsContract.Contacts.CONTENT_URI	content://com.android.contacts/contacts
联系人电话	ContactsContract.CommonDataKinds.Phone.CONTENT_URI	content://com.android.contacts/data/phones
联系人邮箱	ContactsContract.CommonDataKinds.Email.CONTENT_URI	content://com.android.contacts/data/emails
SIM 卡联系人		content://icc/adn
短信	Telephony.Sms.CONTENT_URI	content://sms
彩信	Telephony.Mms.CONTENT_URI	content://mms
通话记录	CallLog.Calls.CONTENT_URI	content://call_log/calls
收件箱	Telephony.Sms.Inbox.CONTENT_URI（短信相关的 URI）	content://sms/inbox
已发送	Telephony.Sms.Sent.CONTENT_URI（短信相关的 URI）	content://sms/sent
草稿箱	Telephony.Sms.Draft.CONTENT_URI（短信相关的 URI）	content://sms/draft
发件箱	Telephony.Sms.Outbox.CONTENT_URI（短信相关的 URI）	content://sms/outbox
发送失败	无	content://sms/failed
待发送列表	无。比如开启飞行模式后，该短信就在待发送列表里	content://sms/queued

13.4 实战项目：音乐播放器——浪花音乐

又到每章结尾的实战项目时间了。手机上的多媒体内容讲究声情并茂、悦目且悦耳，这样才能让用户的感官得到最大享受。影视播放器由于存在视频自身的画面，反而限制了开发者的施展空间；而音乐播放器允许定制播放画面，开发者有足够空间施展拳脚。本章以"音乐播放器——浪花音乐"为压轴实战项目，通过该项目的编码练习巩固和提高开发者的实战技能。

13.4.1 设计思路

大家常见的主流音乐播放器（如 QQ 音乐、酷狗音乐、酷我音乐、虾米音乐、百度音乐等）不外乎有 3 项播放功能：

（1）展示音乐和歌曲列表。

（2）歌曲详情页面滚动展示歌词，并高亮显示当前正在播放的歌词片段。

（3）通过音乐控制条显示播放进度，并提供开始与暂停、拖动播放的功能。

只看文字描述有点抽象，还是先给出播放器的效果图，方便查找对应的功能。图 13-25 所示为播放器的歌曲列表页面，点击顶部的"打开音乐文件"会弹出文件对话框，用于选择音频文件；底部是播放器的控制条，中间为当前手机上的所有音乐文件列表。点击某个音乐项，进入该音乐的详情页面，如图 13-26 所示。页面顶部显示歌曲名称和演唱者，页面底部是播放器



控制条，页面中间为该歌曲对应的歌词内容。

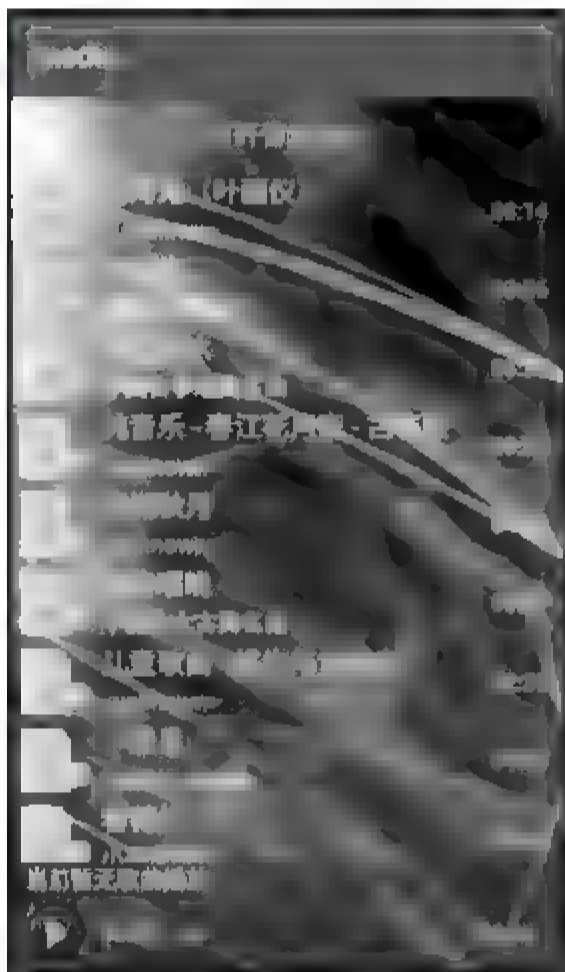


图 13-25 播放器的歌曲列表页面



图 13-26 播放器的歌曲详情页面

接下来对音乐播放器的 3 项功能进行详细剖析。

对于第一点的展示歌曲列表，让用户手动添加不但费时费力，而且用户往往搞不清楚手机上的歌曲都放在哪个目录。我们假设用户是“傻白甜”，开发者做的 App 就得智能贴心，主动帮用户把手机上的歌曲找出来。要想实现这个功能，可以通过内容组件访问系统自带的媒体库，查找并显示媒体库中的歌曲列表。

对于第二点的滚动歌词显示，常见的歌词文件是 LRC 格式的文本文件，内容主要是每句歌词的文字与开始时间。文本文件的解析并不复杂，难点主要是滚动显示。乍看歌词从下往上滚动，适合采用平移动画，然而歌词滚动不是匀速的，因为每句歌词的间隔时间并不固定，只能把整个歌词滚动分解为若干动画，有多少行就有多少个动画。

对于第三点的音乐控制条，总体上使用前面提到的视频控制条。不过音乐控制条更加复杂，因为除了控制音频的播放，还要控制歌词动画的播放。另外，音乐控制条显示在歌曲列表页面上，为了与主流播放器看齐，最好在系统通知栏固定放置音乐控制条。

弄懂了音乐播放器的主要功能，再来看该播放器用到的 App 开发技术。读者能从第一章一直看到本章，学习的耐心真是很好。如果用到前面章节的知识点，这里就一起列举出来。笔者先抛砖引玉，读者发现遗漏的地方可加以补充。

（1）服务 Service：歌曲播放不依赖于某个页面，即使用户回到桌面，歌曲也要继续播放，因此必须在后台服务中播放歌曲。

（2）应用 Application：正在播放的歌曲名称，在播放器的任何页面都能看到，用到了全局内存，要把歌曲名称保存在自定义的 Application 类中。

（3）内容解析器 ContentResolver：系统媒体库中的音频文件，需要通过内容解析器访问媒体库的音频资源，详细路径是 MediaStore.Audio.Media.EXTERNAL_CONTENT_URI。

（4）文件存取：歌词文件与音乐文件在同一个目录下，文件名一样，只是扩展名变为 lrc。

（5）通知 Notify：系统通知栏要显示音乐控制条，就得把后台服务以通知的形式在前台

运行。

（6）媒体播放器 MediaPlayer：播放音频文件，自然会用到媒体播放器。

（7）按键事件 KeyEvent：用户按手机侧面的加、减键，播放器应弹出音量调节对话框，供用户调整音量大小。

（8）动画 Animation：歌词的滚动显示，可使用平移动画，也可使用属性动画实现平移效果。

（9）其余高级控件：如列表视图 ListView、进度条 ProgressBar、拖动条 SeekBar 等，有待读者进一步发掘。

不看不知道，一看吓一跳。如果仅播放声音，技术上只要 Activity 加 MediaPlayer 就行，最多再加一个媒体控制条 MediaController，三板斧够用了。但是要让播放器变得生动活泼，要让用户真正去欣赏音乐，开发者要做的工作就不是实现基础功能，而是从界面设计到用户体验，每个细节都要充分考虑，所以实际运用的技术远远不止三板斧。

13.4.2 小知识：可变字符串 SpannableString

大家都知道，文本控件家族显示文本内容使用 setText 方法，使用 setTextColor 方法设置文本颜色，使用 setTextSize 方法设置文本大小，使用 setTextAppearance 方法设置文本样式（包括颜色、大小、风格等）。普通的用法只能对控件的所有文本做统一设置，如果想对前一段文本加大加粗，对中间一段文本显示红色，再将后面一段文本换成图像，就无能为力了。为了解决分段文本使用不同样式的需求，Android 提供了可变字符串 SpannableString，通过该工具实现对文本分段显示。

SpannableString 的原理是给指定位置的文本赋予对应的样式，从而告知系统这段文本的显示方式。具体到编码有 3 个步骤，说明如下：

步骤 01 从指定文本字符串构造一个 SpannableString 对象。

步骤 02 调用 SpannableString 对象的 setSpan 方法设置指定文本段的显示风格。该方法的第一个参数为风格样式对象，第二个参数为文本段的起始位置，第 3 个参数为文本段的终止位置，第 4 个参数为风格的范围标志，用来标识在文本段前后输入新字符时是否令它们应用这个风格（主要对 EditText 起作用）。风格范围标志的取值说明见表 13-5。

步骤 03 调用文本控件对象的 setText 方法设置定义好的 SpannableString 对象。

表13-5 风格范围标志的取值说明

Spanned 类的范围标志	说明
SPAN_EXCLUSIVE_EXCLUSIVE	前后都不包括
SPAN_INCLUSIVE_EXCLUSIVE	前面包括，后面不包括
SPAN_EXCLUSIVE_INCLUSIVE	前面不包括，后面包括
SPAN_INCLUSIVE_INCLUSIVE	前后都包括

显示风格的定义在 android.text.style 包中，总共有 30 多个。当然，常用的没这么多，笔者整理了 8 个常用的显示风格，详见表 13-6。

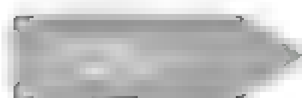


表13-6 常用的显示风格类列表

可变字符串的显示风格类	说明
RelativeSizeSpan	设置文字大小。1.0表示正常大小，0.5表示缩小到原来的一半，2.0表示放大到原来的两倍
StyleSpan	设置文字字体。字体风格的取值说明见表13-7
ForegroundColorSpan	设置文字的颜色
BackgroundColorSpan	设置文字的背景色
UnderlineSpan	给文字加下划线
StrikethroughSpan	给文字加删除线
ImageSpan	把文字替换为图片
URLSpan	给文字添加超链接

表13-7 字体风格的取值说明

Typeface 类的字体风格	说明
NORMAL	正常字体
BOLD	加粗字体
ITALIC	倾斜字体
BOLD_ITALIC	既加粗又设置为斜体

下面是使用 SpannableString 设置文字样式的代码：

```
public class SpannableActivity extends AppCompatActivity {
    private TextView tv_spannable;
    private String mText = "为人民服务";
    private String mKey = "人民";
    private int mBeginPos, mEndPos;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_spannable);
        tv_spannable = (TextView) findViewById(R.id.tv_spannable);
        tv_spannable.setText(mText);
        mBeginPos = mText.indexOf(mKey);
        mEndPos = mBeginPos + mKey.length();
        initSpannableSpinner();
    }

    private void initSpannableSpinner() {
        ArrayAdapter<String> spannableAdapter = new ArrayAdapter<String>(this,
                R.layout.item_select, spannableArray);
        Spinner sp_spannable = (Spinner) findViewById(R.id.sp_spannable);
```

```
        sp_spannable.setPrompt("可变字符串样式：");
        sp_spannable.setAdapter(spannableAdapter);
        sp_spannable.setOnItemSelectedListener(new SpannableSelectedListener());
        sp_spannable.setSelection(0);
    }

    private String[] spannableArray={
            "增大字号", "加粗字体", "前景红色", "背景绿色", "下划线", "表情图片" };
    class SpannableSelectedListener implements OnItemSelectedListener {
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            SpannableString spanText = new SpannableString(mText);
            if (arg2 == 0) {
                spanText.setSpan(new RelativeSizeSpan(1.5f), mBeginPos, mEndPos,
                        Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            } else if (arg2 == 1) {
                spanText.setSpan(new StyleSpan(Typeface.BOLD), mBeginPos, mEndPos,
                        Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            } else if (arg2 == 2) {
                spanText.setSpan(new ForegroundColorSpan(Color.RED), mBeginPos,
                        mEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            } else if (arg2 == 3) {
                spanText.setSpan(new BackgroundColorSpan(Color.GREEN), mBeginPos,
                        mEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            } else if (arg2 == 4) {
                spanText.setSpan(new UnderlineSpan(), mBeginPos, mEndPos,
                        Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            } else if (arg2 == 5) {
                spanText.setSpan(new ImageSpan(SpannableActivity.this, R.drawable.people),
                        mBeginPos, mEndPos, Spanned.SPAN_EXCLUSIVE_EXCLUSIVE);
            }
            tv_spannable.setText(spanText);
        }

        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }
}
```

SpannableString 的不同风格效果如图 13-27～图 13-32 所示。其中，图 13-27 所示为加大字体后的效果，图 13-28 所示为加粗字体后的效果，图 13-29 所示为修改文字颜色后的效果，图 13-30 所示为修改文字背景后的效果，图 13-31 所示为增加下划线后的效果，图 13-32 所示为把文字替换成图片后的效果。

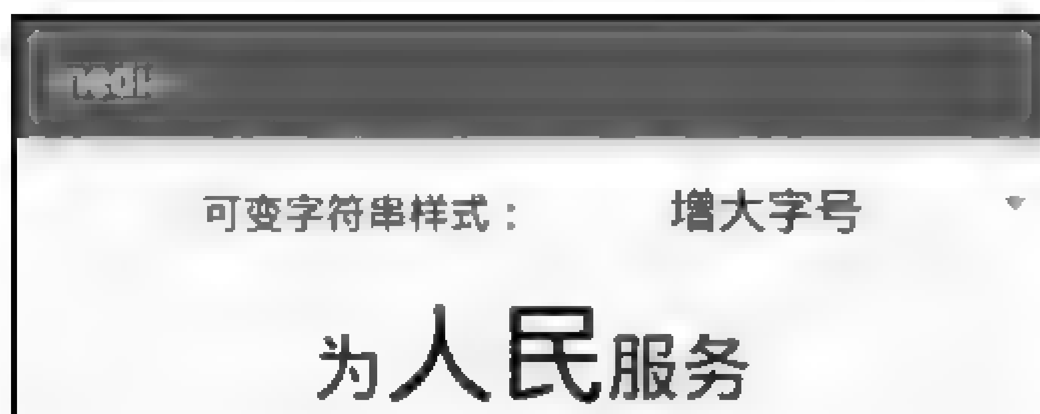


图 13-27　增大字体的风格

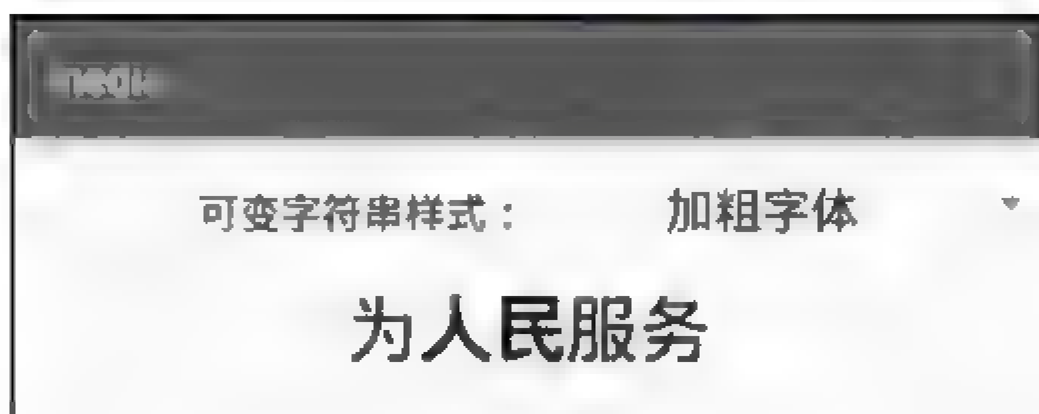


图 13-28　加粗字体的风格

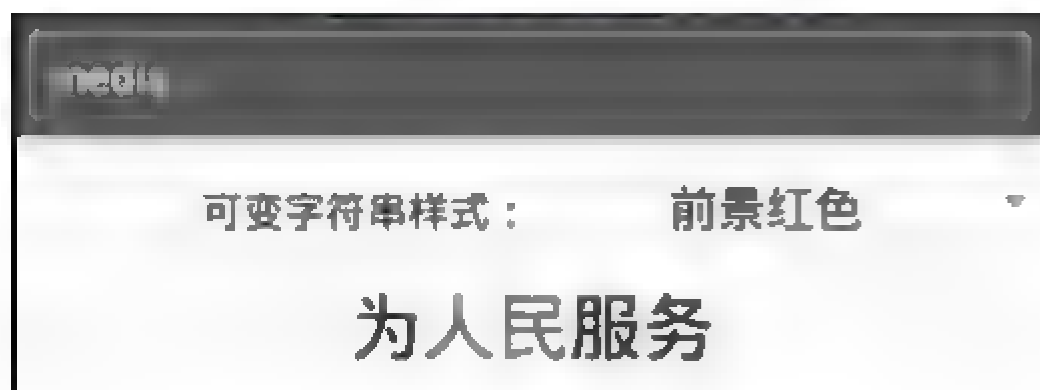


图 13-29　修改文字颜色的风格

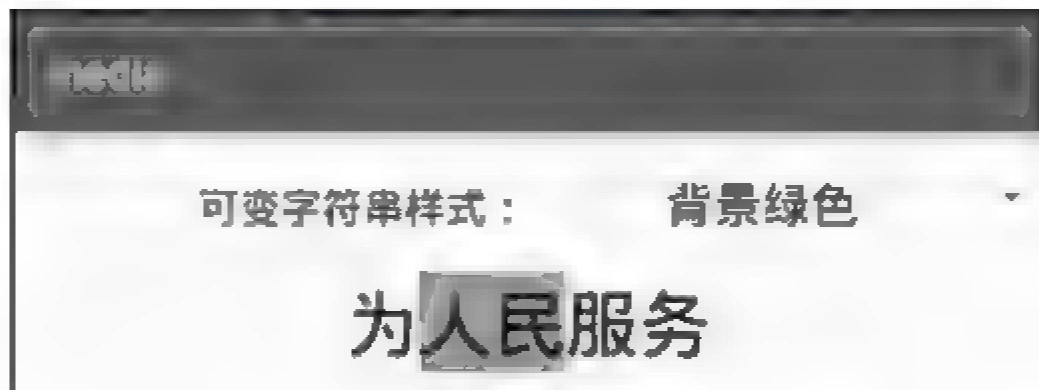


图 13-30　修改文字背景色的风格

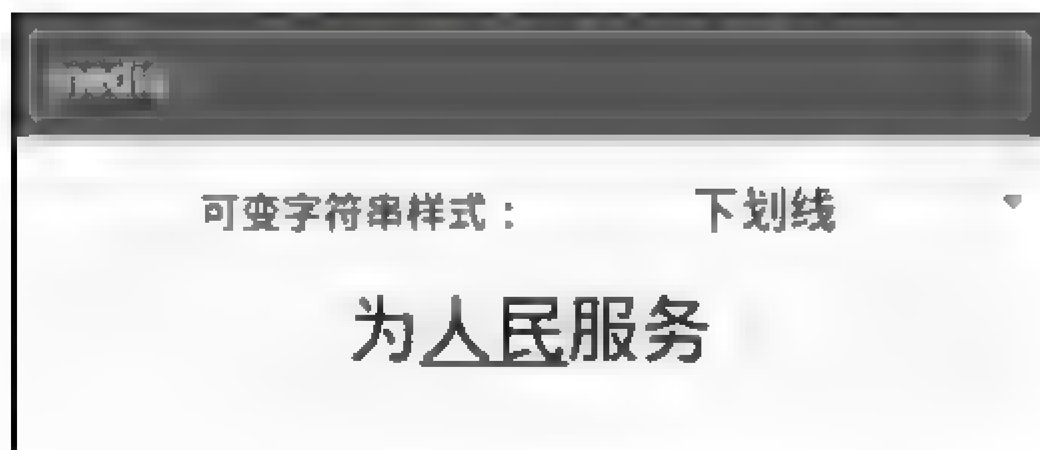


图 13-31　添加下划线的风格



图 13-32　图片替换文字的风格

读者是否对图 13-32 似曾相识？使用 QQ 聊天时会自动把特定字符转成表情图片，比如把文字内容中的“:)”显示为笑脸图片，在 Android 设备上可通过 SpannableString 实现该功能。

13.4.3　代码示例

编码与测试方面需要注意以下 5 点：

（1）如果把动画描述定义在 XML 文件中，动画定义文件就要放在 res/anim 目录下。

（2）打开音乐文件，要记得为 AndroidManifest.xml 添加 SD 卡的权限配置：

```
<!-- SD 卡 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

（3）AndroidManifest.xml 的 application 节点注意补充 android:name=".MainApplication"；另外，注册音乐播放服务的 service，注册代码如下：

```
<service android:name=".service.MusicService" android:enabled="true" />
```

（4）测试设备的 Android 版本要求不低于 Android 4.4.2，因为属性动画的暂停和恢复方法是在 4.4.2 后引入的。

（5）要在真机上测试实战项目，如果在模拟器上测试，就会发现 MP3 标题乱码。这是因为中文歌曲的 MP3 标签采用 GBK 编码，而模拟器采用 UTF8 编码，两者对汉字的编码格式不一致。如果用真机测试，国产机厂商已经帮我们解决了汉字编码问题。

具体的代码编写还存在 3 个技术要点，记录如下：

1. 使用内容解析器 ContentResolver 访问媒体库

音频资源对应的内容路径是 MediaStore.Audio.Media.EXTERNAL_CONTENT_URI，内容解析器通过 query 方法访问该 URI 获得记录游标，还得把详细记录字段逐个读取出来，音频资源的字段信息说明见表 13-8。

表13-8　音频资源的字段信息说明

MediaStore 类的音频资源字段	说明
Audio.Media._ID	歌曲编号
Audio.Media.TITLE	歌曲的标题名称
Audio.Media.ALBUM	歌曲的专辑名称
Audio.Media.DURATION	歌曲的播放时间
Audio.Media.SIZE	歌曲文件的大小
Audio.Media.ARTIST	歌曲的演唱者
Audio.Media.DATA	歌曲文件的完整路径

2. 解析 LRC 歌词文件

简要介绍一下 LRC 文件的内容格式，开发者关心的主要是内部的时间信息与歌词文字。下面是一个 LRC 歌词的片段：

```
[offset:500]
[00:26.53]真情像草原广阔
[00:32.78]层层风雨不能阻隔
[00:38.87]总有云开 日出时候
[00:45.68]万丈阳光照耀你我
[02:26.49][00:51.68]真情像梅花开过
[02:32.68][00:57.94]冷冷冰雪不能掩没
```

歌词第一行有一个 offset 标签，表示歌词标注的时间与音乐文件的时间偏移。歌词行的前面是中括号括起来的时间戳，时间戳的数据格式为“分:秒.毫秒”，表示该行歌词的起始时间。如果某行歌词被演唱多遍，那么歌词文字前面会有多个时间戳。

3. 歌词滚动动画的播放控制

一般动画启动后很快就会结束，但歌词滚动动画不是这样的，用户点击控制条上的暂停按钮，不但播放器要暂停播放，而且歌词要暂停滚动。平移动画 TranslateAnimation 不支持暂停和恢复操作，不止平移动画，所有补间动画都不支持暂停和恢复。难道要自己重定义动画？山穷水尽疑无路，柳暗花明又一村。幸好 Android 提供了属性动画，不但支持所有补间动画效

果，而且支持暂停和恢复操作，还等什么，赶紧把 TranslateAnimation 换成 ObjectAnimator 吧！

现在音乐播放器的编码没什么难点，如果不出状况，读者就能很快看到自己的 App 作品。图 13-33 所示为音乐播放器的效果画面。点击歌曲列表中的歌名《一剪梅》，进入该歌曲的播放界面，歌词文字随着时间流逝缓慢向上滚动，当前演唱的歌词行会高亮显示。播放一段时间后，控制条的进度移到右边，歌词也大半上翻，高亮的歌词行移向后面的文字，如图 13-34 所示。

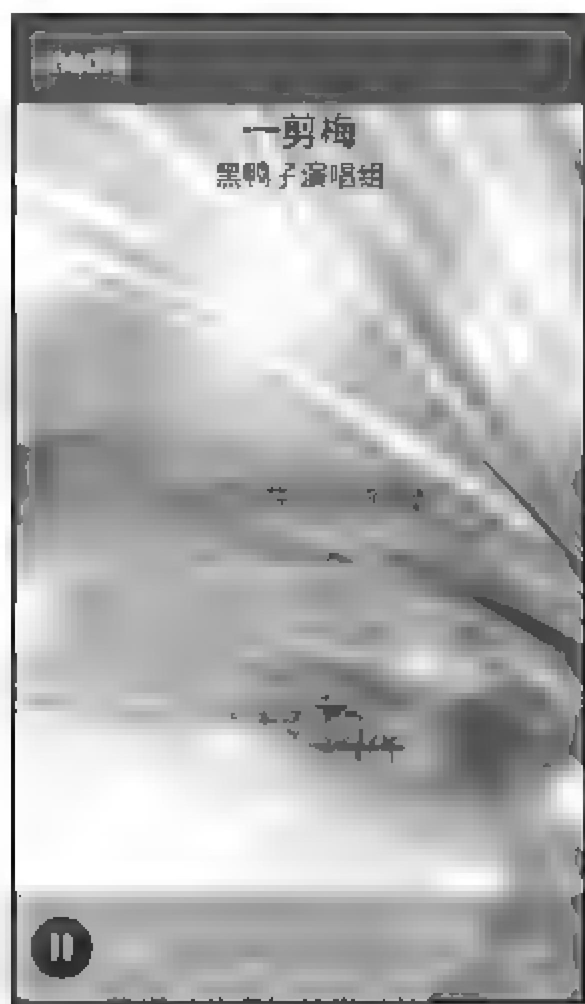


图 13-33　一剪梅开始播放

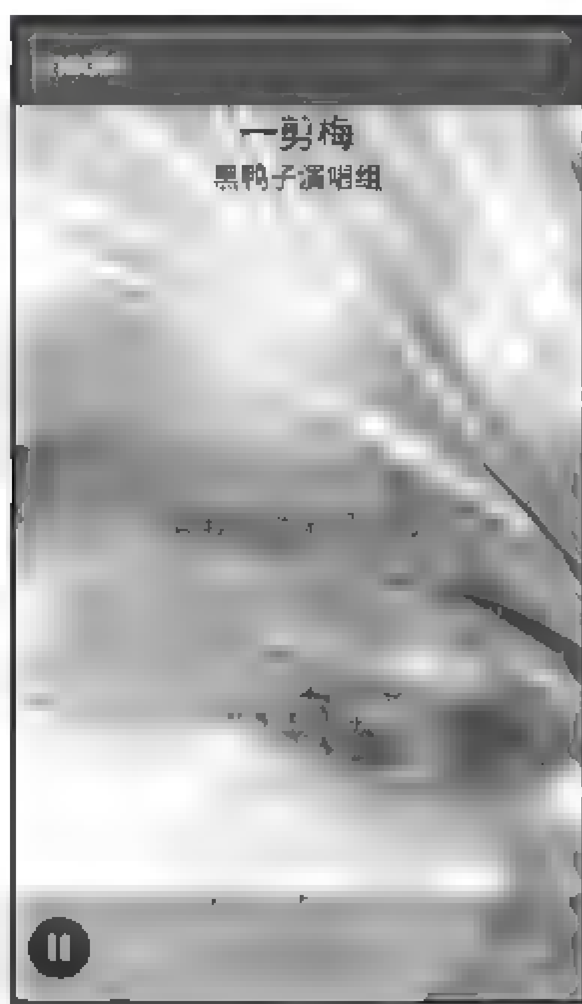


图 13-34　一剪梅正在播放

接着按返回键，后退到歌曲列表页面，页面下方的控制条显示当前的播放进度，时间计数随着歌曲播放而不断刷新，如图 13-35 所示。在歌曲列表页面点击歌名《上海滩》，进入该歌曲的播放界面，此时《一剪梅》停止播放，转为播放《上海滩》，如图 13-36 所示。



图 13-35　回到歌曲列表页面

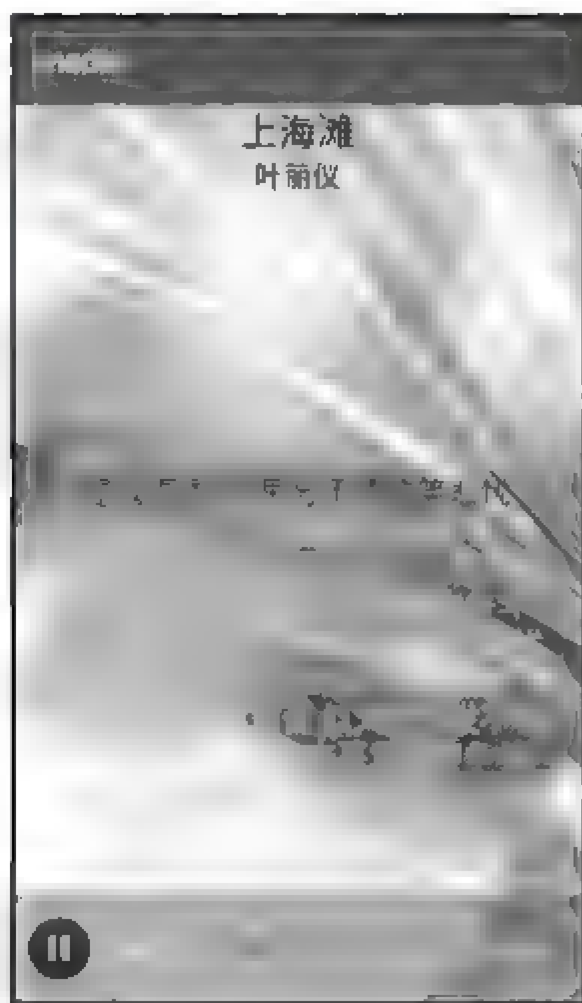


图 13-36　开始播放上海滩

最后下拉系统通知栏，应该能够看到播放器的控制条，如图 13-37 所示。在通知栏上不但可以自动刷新播放进度，而且可以进行暂停和恢复播放的操作。



图 13-37　通知栏上的音乐控制条

下面是音乐播放界面的代码：

```
@TargetApi(Build.VERSION_CODES.KITKAT)
public class MusicDetailActivity extends AppCompatActivity implements
        AnimatorListener, OnSeekChangeListener, VolumeAdjustListener {
    private static final String TAG = "MusicDetailActivity";
    private TextView tv_title;
    private TextView tv_artist;
    private TextView tv_music;
    private MusicInfo mMusic;
    private AudioController ac_play;
    private LyricsLoader mLoader;
    private ArrayList<LrcContent> mLrcList;
    private AudioManager mAudioMgr;
    private VolumeDialog dialog;
    private MainApplication app;
    private Handler mHandler = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_music_detail);
        tv_title = (TextView) findViewById(R.id.tv_title);
        tv_artist = (TextView) findViewById(R.id.tv_artist);
        tv_music = (TextView) findViewById(R.id.tv_music);
        ac_play = (AudioController) findViewById(R.id.ac_play);
        ac_play.setOnSeekChangeListener(this);
        mMusic = getIntent().getParcelableExtra("music");
        tv_title.setText(mMusic.getTitle());
        tv_artist.setText(mMusic.getArtist());
        mLoader = LyricsLoader.getInstance(mMusic.getUrl());
        mLrcList = mLoader.getLrcList();
        mLineHeight = Math.round(MeasureUtil.getTextHeight("好", tv_music.getTextSize()));
        mAudioMgr = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        app = MainApplication.getInstance();
        playMusic();
    }

    @Override
```

```
protected void onDestroy() {
        super.onDestroy();
        mHandler.removeCallbacksAndMessages(null);
}

private int frequence = 8000;
private int channelConfig = AudioFormat.CHANNEL_IN_STEREO;
private int audioFormat = AudioFormat.ENCODING_PCM_16BIT;
// 播放歌曲
private void playMusic() {
        Log.d(TAG, "song="+mMusic.getTitle());
        if (Utils.getExtendName(mMusic.getUrl()).equals("pcm")) {
                ac_play.setVisibility(View.GONE);
                AudioPlayTask playTask = new AudioPlayTask();
                playTask.execute(mMusic.getUrl(), ""+frequence, ""+channelConfig, ""+audioFormat);
        } else {
                //下面处理歌词
                if (mLoader.getLrcList()!=null && mLrcList.size()>0) {
                        mLrcStr = "";
                        for (int i=0; i<mLrcList.size(); i++) {
                                LrcContent item = mLrcList.get(i);
                                mLrcStr = mLrcStr + item.getLrcStr() + "\n";
                        }
                        tv_music.setText(mLrcStr);
                        tv_music.setAnimation(AnimationUtils.loadAnimation(this,R.anim.alpha_music));
                }
                //播放音乐
                if (app.mFilePath==null || !app.mFilePath.equals(mMusic.getUrl())) {
                        Intent intent = new Intent(this, MusicService.class);
                        intent.putExtra("is_play", true);
                        intent.putExtra("music", mMusic);
                        startService(intent);
                        mHandler.postDelayed(mRefreshLrc, 150);
                } else {
                        onMusicSeek(0, app.mMediaPlayer.getCurrentPosition());
                }
                mHandler.postDelayed(mRefreshCtrl, 100);
        }
}

//刷新进度条
private Runnable mRefreshCtrl = new Runnable() {
        @Override
```

```
        public void run() {
            ac_play.setCurrentTime(app.mMediaPlayer.getCurrentPosition(), 0);
            if (app.mMediaPlayer.getCurrentPosition() >= app.mMediaPlayer.getDuration()) {
                ac_play.setCurrentTime(0, 0);
            }
            mHandler.postDelayed(this, 500);
        }
    };

    private int mCount = 0;
    private float mCurrentHeight = 0;
    private float mLineHeight = 0;
    //计算每行歌词的动画
    private Runnable mRefreshLrc = new Runnable() {
        @Override
        public void run() {
            if (mLoader.getLrcList()==null || mLrcList.size()<=0) {
                return;
            }
            int offset = mLrcList.get(mCount).getLrcTime()
                    - ((mCount==0)?0:mLrcList.get(mCount-1).getLrcTime()) - 50;
            if (offset <= 0) {
                return;
            }
            startAnimation(mCurrentHeight - mLineHeight, offset);
        }
    };

    private int mPrePos = -1, mNextPos = 0;
    private String mLrcStr;
    private ObjectAnimator animTranY;
    //歌词滚动动画
    public void startAnimation(float aimHeight, int offset) {
        animTranY = ObjectAnimator.ofFloat(tv_music, "translationY", mCurrentHeight, aimHeight);
        animTranY.setDuration(offset);
        animTranY.setRepeatCount(0);
        animTranY.addListener(this);
        animTranY.start();
        mCurrentHeight = aimHeight;
        if (app.mMediaPlayer.isPlaying() != true) {
            mHandler.postDelayed(new Runnable() {
                @Override
                public void run() {
```

```
                    animTranY.pause();
                }
            }, offset+100);
        }
    }

    @Override
    public void onAnimationStart(Animator animation) {
    }

    @Override
    public void onAnimationEnd(Animator animation) {
        if (mCount < mLrcList.size()) {
            mNextPos = mLrcStr.indexOf("\n", mPrePos+1);
            SpannableString spanText = new SpannableString(mLrcStr);
            spanText.setSpan(new ForegroundColorSpan(Color.RED), mPrePos+1,
                    mNextPos>0?mNextPos:mLrcStr.length()-1, Spanned.SPAN_EXCLUSIVE_
EXCLUSIVE);
            mCount++;
            tv_music.setText(spanText);
            if (mNextPos > 0 && mNextPos < mLrcStr.length()-1) {
                mPrePos = mLrcStr.indexOf("\n", mNextPos);
                mHandler.postDelayed(mRefreshLrc, 50);
            }
        }
    }

    @Override
    public void onAnimationCancel(Animator animation) {
    }

    @Override
    public void onAnimationRepeat(Animator animation) {
    }

    //音乐控制条的拖动操作
    @Override
    public void onMusicSeek(int current, int seekto) {
        Log.d(TAG, "current="+current+", seekto="+seekto);
        if (animTranY != null) {
            animTranY.cancel();
        }
        mHandler.removeCallbacks(mRefreshLrc);
```

```
        int i;
        for (i=0; i<mLrcList.size(); i++) {
            LrcContent item = mLrcList.get(i);
            if (item.getLrcTime() > seekto) {
                break;
            }
        }
        mCount = i;
        mPrePos = -1;
        mNextPos = 0;
        if (mCount > 0) {
            for (int j = 0; j < mCount; j++) {
                mNextPos = mLrcStr.indexOf("\n", mPrePos + 1);
                mPrePos = mLrcStr.indexOf("\n", mNextPos);
            }
        }
        startAnimation(-mLineHeight*i, 100);
    }

    @Override
    public void onMusicPause() {
        animTranY.pause();
    }

    @Override
    public void onMusicResume() {
        animTranY.resume();
    }

    //音量调节对话框
    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_VOLUME_UP) {
            showVolumeDialog(AudioManager.ADJUST_RAISE);
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN) {
            showVolumeDialog(AudioManager.ADJUST_LOWER);
            return true;
        } else if (keyCode == KeyEvent.KEYCODE_BACK) {
            finish();
        }
        return false;
    }
```

```
        private void showVolumeDialog(int direction) {
            if (dialog==null || dialog.isShowing()!=true) {
                dialog = new VolumeDialog(this);
                dialog.setVolumeAdjustListener(this);
                dialog.show();
            }
            dialog.adjustVolume(direction, true);
            onVolumeAdjust(mAudioMgr.getStreamVolume(AudioManager.STREAM_MUSIC));
        }

        @Override
        public void onVolumeAdjust(int volume) {
        }
    }
```

13.5 小　　结

本章主要介绍App开发用到的常见多媒体技术，包括自定义相册的实现（画廊、图像切换器、卡片视图、调色板）、影视播放器的实现（视频视图、媒体控制条、阶段实战项目“爱看剧场”）、ContentProvider内容组件的用法（内容提供器、内容解析器、内容操作器、内容观察器）。最后设计了一个实战项目“音乐播放器——浪花音乐”，在该项目的App编码中采用了本书到目前为止的主要技术点，实现了歌曲的播放控制和歌词的滚动显示。另外，介绍了可变字符串的种类及其使用说明。

通过本章的学习，读者应该能够掌握以下5种开发技能：

（1）学会如何使用图像控件实现自定义相册。

（2）学会如何使用视频控件实现影视播放器。

（3）学会如何使用音频控件实现音乐播放器。

（4）学会ContentProvider组件的用法，如封装数据的对外接口，对开放内容接口的系统数据进行查询、修改和监视操作。

（5）学会借助可变字符串在一段文本中运用不同的风格样式。



融合技术

本章介绍融合技术的几个方向，主要包括使用网页集成技术实现不同终端显示同一个网页、使用 JNI 开发技术实现不同平台运行同一套代码、使用局域网共享技术实现不同设备分享同一份文件。最后结合本章所学的知识演示一个实战项目“WIFI 共享器”的设计与实现。

14.1 网页集成

本节介绍融合技术的一个重要方向——网页集成，Web页面可以直接在Android、iOS、Windows等终端上显示，能够减少重复的适配工作，有效降低开发成本。本节首先说明如何使用资产管理器打开文本文件、图片文件以及加载网页，接着逐步阐述网页视图的详细用法，最后利用网页视图实现一个简单浏览器。

14.1.1 资产管理器 AssetManager

如同所有的应用程序那样，App运行时也要读取事先定义好的配置信息，并加载图片等资源文件。一般情况下，这些配置信息与资源文件可以放在工程的res目录中，举例如下：

（1）图片文件与图形定义文件可以放在res/drawable目录。
（2）字符串定义可以放在res/values/strings.xml文件中。
（3）颜色值定义可以放在res/values/colors.xml文件中。
（4）整型数定义可以放在res/values/integers.xml文件中。
（5）各类数组定义可以放在res/values/arrays.xml文件中。
（6）音频等其他二进制流文件可以放在res/raw目录。

乍看之下，res目录已经允许保存几乎所有配置信息与资源文件了，不过事情往往存在各种预料之外的情况，比如以下业务场景就无法使用res配置：

（1）大批量的初始化数据，需要在App第一次安装时导入数据库。因为res/values目录下放的是键值对数据（如key-value），难以转换为数据库中存储的关系型数据。
（2）工程源码要导出为JAR包，作为一个SDK给其他工程使用。因为res目录无法集成到jar包中，所以待集成的图片资源不可放在res目录。
（3）如网页HTML这种需要保持原有格式的文件，不适合放在res目录中进行编译。

基于此，Android提供了一个assets目录用来保存以上特殊需求的文件。在Android Studio中创建一个新模块，默认没有assets目录，开发者得自己在src/main目录下新建assets目录，然后在该目录中存放各种要求保持原有格式的文件。

因为assets目录下的资产文件不会被系统编译，所以无法通过R.*.*这种方式访问，需要使用另外的工具——资产管理器AssetManager访问。通过该工具，我们能够以输入流方式打开assets目录的文件，并将输入流转换为文本或图像。

在Activity代码中调用getAssets方法可获得AssetManager对象。下面是AssetManager的常用方法说明。

- list：列出指定目录下的文件与文件夹列表数组。
- open：打开资产文件，返回输入流InputStream对象。访问模式默认是AssetManager.ACCESS_STREAMING，表示流式访问，即顺序读取。

- close：关闭资产管理器。

assets 目录保存的多是文本文件与图片文件。使用 AssetManager 读取文本和图像的代码如下：

```
public static String getTxtFromAssets(Context context, String fileName) {
    String result = "";
    try {
        InputStream is = context.getAssets().open(fileName);
        int lenght = is.available();
        byte[]  buffer = new byte[lenght];
        is.read(buffer);
        result = new String(buffer, "utf8");
    } catch (Exception e) {
        e.printStackTrace();
    }
    return result;
}

public static Bitmap getImgFromAssets(Context context, String fileName) {
    Bitmap bitmap = null;
    try {
        InputStream is = context.getAssets().open(fileName);
        bitmap = BitmapFactory.decodeStream(is);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return bitmap;
}
```

资产管理器读取文本与图像的效果如图 14-1 和图 14-2 所示。其中，图 14-1 所示为从 assets 目录读取并显示文本文件的画面，图 14-2 所示为从 assets 目录读取并显示图片文件的画面。



图 14-1　从资产目录读取文本



图 14-2　从资产目录读取图片

14.1.2　网页视图 WebView

前面提到 assets 目录可保存网页文件，由于网页不是一般的文本文件，而是包含一系列



html 标签的页面描述定义，因此如果想显示网页的效果画面而非源代码，就得借助于网页视图 WebView。WebView 相当于 Android 的一个浏览器内核，可内嵌并展示 Web 页面，并处理 App 与 Web 的交互操作。

调用 WebView 对象的 loadUrl 方法可让网页视图显示资产目录中的网页，注意要在网页路径前加上 file:///android_asset/，表示该网页来自于本地的 assets 目录，具体代码如下：

```
public class WebLocalActivity extends AppCompatActivity {
    private String mFilePath = "file:///android_asset/html/index.html";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_web_local);
        TextView tv_web_path = (TextView) findViewById(R.id.tv_web_path);
        WebView wv_assets_web = (WebView) findViewById(R.id.wv_assets_web);
        tv_web_path.setText("下面网页来源于资产文件"+mFilePath);
        wv_assets_web.loadUrl(mFilePath);
        wv_assets_web.setWebViewClient(new WebViewClient());
    }
}
```

WebView 展示本地网页的效果如图 14-3 所示。页面左边是图片，右边是诗歌的文本。



图 14-3 从资产目录读取网页

网页视图可以访问本地网页，也可以访问外部网页。在电脑浏览器上查看网页时经常通过点击超链接打开新窗口。在手机上，App 要实现超链接跳转，可参照第 13 章的可变字符串 UrlSpan，该风格把指定位置的文字转为超链接，点击超链接文字即可跳转到相应 URL。注意这里的跳转 URL 其实是在一个网页视图中打开的。

看来 App 针对超链接的处理比 HTML 复杂，虽然复杂了点，但是套用固定的代码模板使用也不难。使用超链接风格打开网页视图的代码如下：

```
    private void showUrlSpan() {
        SpannableString spanText = new SpannableString(mText);
        //调用 setMovementMethod 方法设置 LinkMovementMethod 后，点击超链接才有反应
        tv_spannable.setMovementMethod(LinkMovementMethod.getInstance());
        Spannable sp = (Spannable) Html.fromHtml("<a href=\"\">"+mKey+"</a>");
        CharSequence text = sp.toString();
        URLSpan[] urls = sp.getSpans(0, text.length(), URLSpan.class);
        for (URLSpan url : urls) {
            MyURLSpan myURLSpan = new MyURLSpan(url.getURL());
            spanText.setSpan(myURLSpan, mBeginPos, mEndPos, Spanned.SPAN_EXCLUSIVE_
EXCLUSIVE);
```

```
        }
        tv_spannable.setText(spanText);
    }

    private class MyURLSpan extends URLSpan {
        public MyURLSpan(String url) {
            super(url);
        }

        @Override
        public void onClick(View widget) {
            wv_spannable.setVisibility(View.VISIBLE);
            wv_spannable.loadUrl("http://blog.csdn.net/aqi00");
            wv_spannable.requestFocus();
            wv_spannable.setWebViewClient(new WebViewClient());
            return;
        }
    }
```

超链接风格的文字效果如图 14-4 所示。文字加了下划线，并且文字与下划线都高亮显示。点击超链接后，在网页视图中打开指定的 URL 地址，显示的 Web 页面如图 14-5 所示，看起来是手机版的网页。

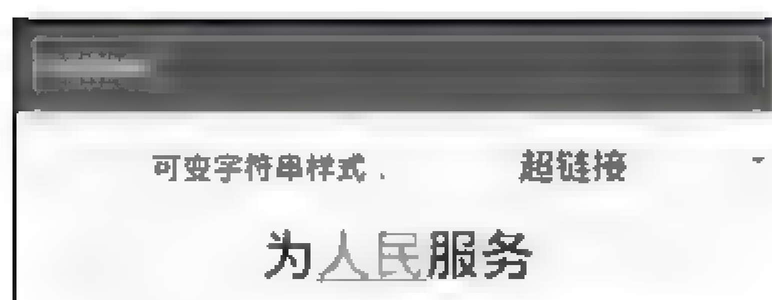


图 14-4　超链风格的文字效果



图 14-5　点击超链接打开网页

14.1.3　简单浏览器

注意前面使用的 WebView，除了调用 loadUrl 方法外，还调用了其他方法（如 setWebViewClient 等）。下面说明 WebView 的常用方法。

- loadUrl：加载指定的 URL，URL 可以是 http 打头的外部网址，也可以是 file 打头的资产网页。
- getSettings：获取浏览器的网页设置信息。返回一个网页设置 WebSettings 对象。
- addJavascriptInterface：添加供 JavaScript 调用的 App 接口。

- setWebViewClient：设置网页视图的客户端对象 WebViewClient，如果已调用 loadUrl 方法，就必须同时调用本方法。
- setWebChromeClient：设置浏览器的网页交互客户端 WebChromeClient。
- setDownloadListener：设置文件下载监听器 DownloadListener。
- loadData：加载文本数据。第二个参数表示媒体类型，如 text/html；第三个参数表示数据的编码格式，如 base64 表示采用 BASE64 编码，其余值（包括 null）表示 URL 编码。
- canGoBack：判断页面能否返回。
- goBack：返回上一个页面。
- canGoForward：判断页面能否前进。
- goForward：前进到下一个页面。
- reload：重新加载页面。
- stopLoading：停止加载页面。

上述方法中有 4 个组件需要补充描述，包括网页设置 WebSettings、网页视图客户端 WebViewClient、网页交互客户端 WebChromeClient 和文件下载监听器 DownloadListener。

1. 网页设置 WebSettings

WebSettings 用于管理网页视图的加载属性，指明了什么该做、什么不该做。调用 WebView 对象的 getSettings 方法即可获得 WebSettings 对象。下面是 WebSettings 的常用设置方法。

以下是基本的加载设置。

- setLoadsImagesAutomatically：设置是否自动加载图片。如果设置为 false，就表示无图模式。
- setDefaultTextEncodingName：设置默认的文本编码，如 UTF-8、GBK 等。
- setJavaScriptEnabled：设置是否支持 JavaScript。
- setJavaScriptCanOpenWindowsAutomatically：设置是否允许 JavaScript 自动打开新窗口，即 JS 的 window.open 方法是否适用。

以下是与网页适配有关的设置。

- setSupportZoom：设置是否支持页面缩放。
- setBuiltInZoomControls：设置是否出现缩放工具。
- setUseWideViewPort：当容器超过页面大小时，是否将页面放大到塞满容器宽度的尺寸。
- setLoadWithOverviewMode：当页面超过容器大小时，是否将页面缩小到容器能够装下的尺寸。
- setLayoutAlgorithm：设置自适应屏幕的算法，一般是 LayoutAlgorithm.SINGLE_COLUMN。如果不设置，Android4.2.2 及之前的版本就可能出现表格错乱的情况。

以下是与存储有关的设置。

- setAppCacheEnabled：设置是否启用 App 缓存。
- setAppCachePath：设置 App 缓存文件的路径。
- setAllowFileAccess：设置是否允许访问文件，如 WebView 访问 SD 卡的文件。

- setDatabaseEnabled：设置是否启用数据库。
- setDomStorageEnabled：设置是否启用本地存储。
- setCacheMode：设置使用的缓存模式。缓存模式的取值说明见表 14-1。

表14-1 缓存模式的取值说明

WebSettings 类的缓存模式	说明
LOAD_CACHE_ELSE_NETWORK	优先使用缓存
LOAD_NO_CACHE	不使用缓存
LOAD_CACHE_ONLY	只使用缓存

2. 网页视图客户端 WebViewClient

可以将 WebViewClient 看作网页加载监听器，用于处理与加载动作有关的事件，WebView 对象调用 setWebViewClient 方法即可设置客户端。需要重写以下方法说明。

- onPageStarted：页面开始加载时触发。可在此弹出进度对话框 ProgressFialog。
- onPageFinished：页面加载结束时触发。可在此关闭进度对话框。
- onReceivedError：收到错误信息时触发。
- onReceivedSslError：收到 SSL 错误时触发。
- shouldOverrideUrlLoading：重写该方法的目的是判断每当点击网页里中链接时，是想在当前的网页视图里跳转还是跳转到系统自带的浏览器。

在当前的网页视图内部跳转，重写方法代码如下：

```
public boolean shouldOverrideUrlLoading(WebView view, String url) {
    view.loadUrl(url);
    return true;
}
```

3. 网页交互客户端 WebChromeClient

WebChromeClient 用于处理网页与 App 之间的交互事件，WebView 对象调用 setWebChromeClient 方法即可设置客户端。WebChromeClient 需要重写的方法说明如下：

- onReceivedTitle：收到页面标题时触发。
- onProgressChanged：页面加载进度发生变化时触发。可在此刷新进度对话框的进度条。
- onJsAlert：网页的 JS 代码调用 alert 方法时触发。可在此弹出自定义的提示对话框。
- onJsConfirm：网页的 JS 代码调用 confirm 方法时触发。可在此弹出自定义的确认对话框。
- onJsPrompt：网页的 JS 代码调用 prompt 方法时触发。可在此弹出自定义的提示对话框。
- onGeolocationPermissionsShowPrompt：网页请求定位权限时触发。可在此弹出一个确认对话框，提示用户是否允许网页获得定位权限。如果不想出现弹窗就允许网页获得权限，重写方法代码如下：

```
public void onGeolocationPermissionsShowPrompt(String origin, Callback callback) {
    callback.invoke(origin, true, false);
```



```
        super.onGeolocationPermissionsShowPrompt(origin, callback);
    }
```

4. 文件下载监听器 DownloadListener

DownloadListener 用于监听网页的下载事件，WebView 对象调用 setDownloadListener 方法即可设置下载监听器。DownloadListener 只有 onDownloadStart 方法需要重写。

- onDownloadStart: 文件开始下载触发。可在此接管下载动作，比如设置文件下载的方式、文件的保存路径等。

了解网页视图相关组件的具体用法后，接下来让我们实现一个简单的浏览器，进一步加深对 WebView 运用的理解。下面是使用 WebView 实现简单浏览器的代码：

```
public class WebBrowserActivity extends AppCompatActivity implements OnClickListener {
    private final static String TAG = "WebBrowserActivity";
    private EditText et_web_url;
    private WebView wv_web;
    private ProgressDialog m_pd;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_web_browser);
        et_web_url = (EditText) findViewById(R.id.et_web_url);
        et_web_url.setText("news.qq.com/");
        wv_web = (WebView) findViewById(R.id.wv_web);
        findViewById(R.id.btn_web_go).setOnClickListener(this);
        findViewById(R.id.ib_back).setOnClickListener(this);
        findViewById(R.id.ib_forward).setOnClickListener(this);
        findViewById(R.id.ib_refresh).setOnClickListener(this);
        findViewById(R.id.ib_close).setOnClickListener(this);
        initWebViewSettings();
    }

    @SuppressLint("SetJavaScriptEnabled")
    private void initWebViewSettings() {
        WebSettings settings = wv_web.getSettings();
        settings.setLoadsImagesAutomatically(true);
        settings.setDefaultTextEncodingName("utf-8");
        settings.setJavaScriptEnabled(true);
        settings.setJavaScriptCanOpenWindowsAutomatically(false);
        settings.setSupportZoom(true);
        settings.setBuiltInZoomControls(true);
        settings.setUseWideViewPort(true);
```

```
            settings.setLoadWithOverviewMode(true);
            settings.setLayoutAlgorithm(LayoutAlgorithm.SINGLE_COLUMN);
        }

        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.btn_web_go) {
                InputMethodManager imm = (InputMethodManager) getSystemService(Context.INPUT_
METHOD_SERVICE);
                imm.hideSoftInputFromWindow(et_web_url.getWindowToken(), 0);
                String url = "http://" + et_web_url.getText().toString();
                Log.d(TAG, "url="+url);
                wv_web.loadUrl(url);
                wv_web.setWebViewClient(mWebViewClient);
                wv_web.setWebChromeClient(mWebChrome);
                wv_web.setDownloadListener(mDownloadListener);
            } else if (v.getId() == R.id.ib_back) {
                if (wv_web.canGoBack()) {
                    wv_web.goBack();
                } else {
                    Toast.makeText(this, "已经是最后一页了", Toast.LENGTH_SHORT).show();
                }
            } else if (v.getId() == R.id.ib_forward) {
                if (wv_web.canGoForward()) {
                    wv_web.goForward();
                } else {
                    Toast.makeText(this, "已经是最前一页了", Toast.LENGTH_SHORT).show();
                }
            } else if (v.getId() == R.id.ib_refresh) {
                //重新加载。停止加载用 stopLoading
                wv_web.reload();
            } else if (v.getId() == R.id.ib_close) {
                finish();
            }
        }

        @Override
        public void onBackPressed() {
            if (wv_web.canGoBack()) {
                wv_web.goBack();
                return;
            } else {
                finish();
```

```
            }
        }

        private WebViewClient mWebViewClient = new WebViewClient() {
            @Override
            public void onReceivedSslError(WebView view, android.webkit.SslErrorHandler handler,
                    android.net.http.SslError error) {
                handler.proceed();
            };

            @Override
            public void onPageStarted(WebView view, String url, Bitmap favicon) {
                super.onPageStarted(view, url, favicon);
                Log.d(TAG, "onPageStarted:" + url);
                if (m_pd == null || m_pd.isShowing() == false) {
                    m_pd = new ProgressDialog(WebBrowserActivity.this);
                    m_pd.setTitle("稍等");
                    m_pd.setMessage("页面加载中……");
                    m_pd.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                    m_pd.show();
                }
            }

            @Override
            public void onPageFinished(WebView view, String url) {
                super.onPageFinished(view, url);
                Log.d(TAG, "onPageFinished:" + url);
                if (m_pd != null && m_pd.isShowing() == true) {
                    m_pd.dismiss();
                }
            }

            @Override
            public void onReceivedError(WebView view, int errorCode, String description, String failingUrl) {
                super.onReceivedError(view, errorCode, description, failingUrl);
                Log.d(TAG, "onReceivedError: url=" + failingUrl+", errorCode="+errorCode+",
description="+description);
                if (m_pd != null && m_pd.isShowing() == true) {
                    m_pd.dismiss();
                }
                Toast.makeText(WebBrowserActivity.this, "页面加载失败，请稍候再试",
Toast.LENGTH_LONG).show();
            }
```

```
            @Override
            public boolean shouldOverrideUrlLoading(WebView view, String url) {
                view.loadUrl(url);
                return true;
            }
        };

        private WebChromeClient mWebChrome = new WebChromeClient() {
            @Override
            public void onProgressChanged(WebView view, int progress) {
                if (m_pd != null && m_pd.isShowing() == true) {
                    m_pd.setProgress(progress);
                }
            }

            @Override
            public void onGeolocationPermissionsShowPrompt(String origin, Callback callback) {
                callback.invoke(origin, true, false);
                super.onGeolocationPermissionsShowPrompt(origin, callback);
            }
        };

        private DownloadListener mDownloadListener = new DownloadListener() {
            @Override
            public void onDownloadStart(String url, String userAgent, String contentDisposition,
                        String mimetype, long contentLength) {
                //此处操作文件下载
            }
        };
    }
```

简单浏览器的展示效果如图 14-6～图 14-9 所示。其中，图 14-6 所示为打开浏览器的初始页面，页面上部为地址栏，下部为控制栏（从左到右依次是前进、后退、刷新、退出等按钮）；在地址栏输入网址并点击“快去”按钮，浏览器显示正在加载的进度对话框，如图 14-7 所示；网页加载完毕后，进度对话框消失，浏览器主视图中显示该网址的 Web 页面，如图 14-8 所示；点击该页面的第一条新闻，浏览器打开该新闻的详情页面，如图 14-9 所示。

要想在前后网页中切换，可点击下方控制栏的前进或后退按钮；要想重新加载当前网页，可点击控制栏的刷新按钮；要想退出浏览器，可点击控制栏右边的退出按钮。读者若有兴趣，也可加入其他高级功能，如设置默认主页、开启无图模式、添加书签管理等内容。



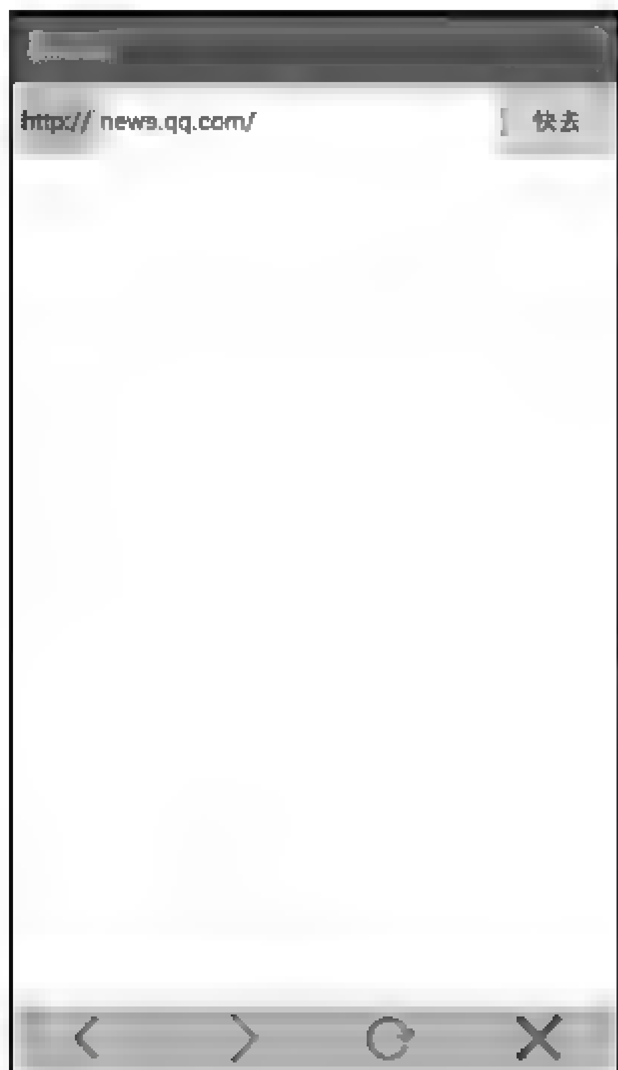


图 14-6　浏览器的初始界面



图 14-7　浏览器加载网页中



图 14-8　浏览器加载网页完成



图 14-9　点击进入焦点新闻

14.2　JNI 开发

本节介绍融合技术的一个重要方向——JNI 开发。C/C++语言具有跨平台特性，苹果操作系统能够直接运行 C/C++代码，如果功能采用 C/C++实现，就很容易在不同平台（如 Android 与 iOS）之间移植。本节首先说明如何在 Android Studio 中搭建 NDK 编译环境；接着阐述如何使用 JNI 接口完成 Java 代码对 C 代码的调用；最后描述 JNI 技术适用的业务场景，并给出一个实际需求的应用项目“JNI 实现加解密”。

14.2.1 NDK 环境搭建

完整的 Android Studio 环境包括 3 个开发工具，即 JDK、SDK 和 NDK，早在第一章就对这些工具做了介绍，这里不妨复习一下。

（1）JDK是Java语言的编译器，因为App采用Java语言开发，所以开发机上要先安装JDK。

（2）SDK 是 Android 应用的编译器，提供了 Android 内核的公共 API 调用，所以开发 App 必须安装 SDK。

（3）NDK 是 C/C++代码的编译器，如果 App 未使用 JNI 技术，就无须安装 NDK；如果 App 用到 JNI，就必须安装 NDK。

NDK 允许开发者在 App 中通过 C/C++代码执行部分操作，然后由 Java 代码通过 JNI 接口调用 C/C++代码。既然本节讲的是 JNI 开发，那么肯定要给 Android Studio 安装 NDK。

下面是 NDK 环境的搭建步骤说明。

01 到谷歌开发者网站下载最新的 NDK 开发包，下载页面地址是 https://developer.android.google.cn/ndk/downloads/index.html。下载完毕后，解压到本地路径，比如笔者把 NDK 解压到了 D:\android-ndk-r13b。注意目录名称不要有中文。

02 在系统中增加 NDK 的环境变量定义，如变量名为 NDK_ROOT，变量值为 D:\android-ndk-r13b。另外，在 Path 变量值后面补充;%NDK_ROOT%。

03 在项目名称上右击，然后在弹出的菜单项中选择 Open Module Settings，打开设置页面，如图 14-10 所示。也可依次选择菜单 File→Project Structure 打开设置页面。

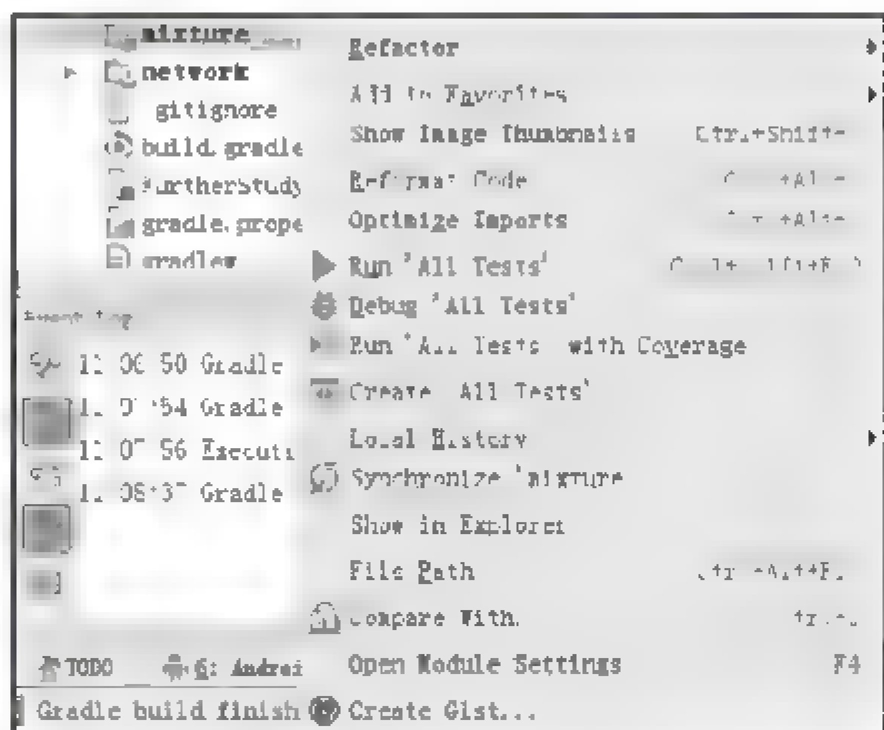


图 14-10 设置页面

在打开的设置页面中依次找到 SDK Location→NDK Location，设置前面解压的 NDK 目录路径，然后单击 OK 按钮，设置页面如图 14-11 所示。

04 在模块的 src/main 路径下创建名为 jni 的目录，h 文件、c 文件、cpp 文件、mk 编译文件都放在该目录下。jni 目录的结构如图 14-12 所示，可以看到 jni 与 java、res 等目录平级。

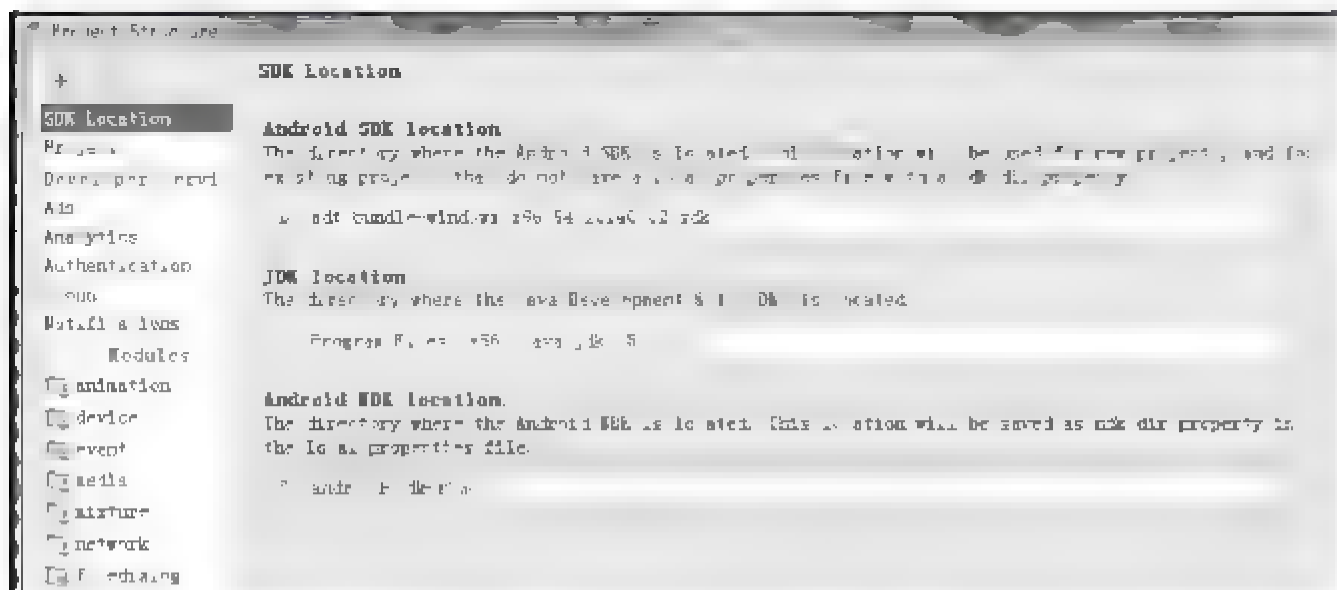


图 14-11 项目结构页面设置 NDK 的安装路径

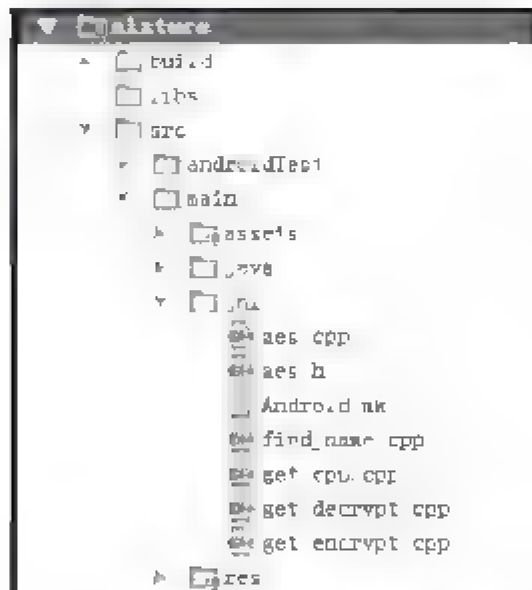


图 14-12 jni 目录在模块工程中的位置

05 打开模块的编译配置文件 build.gradle，在 defaultConfig 节点内补充如下编译配置：

```
ndk {
    moduleName "jni_mix"
    cFlags "-fexceptions"
    ldLibs "log", "z", "m"
    stl "stlport_static"
    abiFilters "armeabi", "armeabi-v7a", "x86"
}
```

在 Android Studio 2.2 及之后版本中，也可直接在 android 节点下补充 mk 文件路径，表示采用外部文件的编译规则，配置说明举例如下：

```
externalNativeBuild {
    ndkBuild {
        path file("src\\main\\jni\\Android.mk")
    }
}
```

06 打开项目的 gradle.properties，增加一行配置：android.useDeprecatedNdk=true。

07 依次选择菜单 Build→Rebuild Project，或选择 Build→Make Module ***，执行 C/C++代码的编译工作。编译通过后，可在“模块名称\build\intermediates\ndk\debug\lib”路径下找到生成的 so 库文件，so 文件的完整路径结构如图 14-13 所示。

08 在 src/main 路径下创建 so 库的保存目录，目录名称为 jniLibs，并将生成的 so 文件复制到该目录下。复制完 so 库的目录结构如图 14-14 所示，可见 jniLibs 与 jni 目录平级。

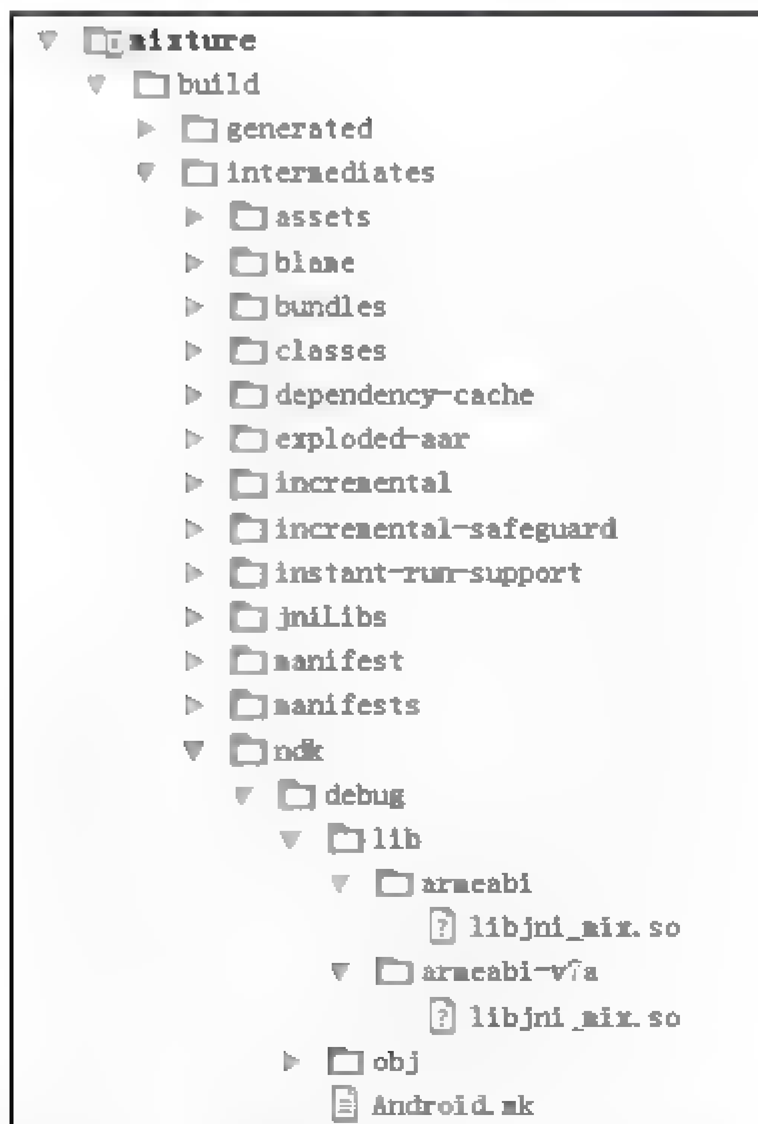


图 14-13 编译生成的 so 文件的路径

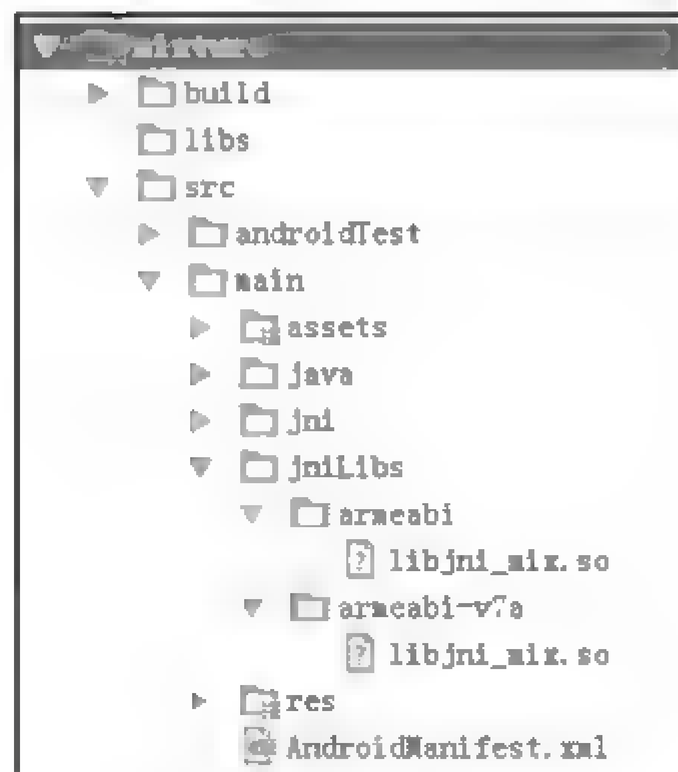


图 14-14 jniLibs 目录在模块工程中的位置

步骤 09 重新运行 App 或重新生成签名 Apk，最后产生的 App 就是封装好 so 库版本。

14.2.2 创建 JNI 接口

JNI 是 Java Native Interface 的缩写，提供了若干 API 实现 Java 和其他语言的通信（主要是 C/C++）。虽然 JNI 是 Java 平台的标准，但是要想在 Android 上使用 JNI，还得配合 NDK 才行。NDK 提供了 C/C++标准库的头文件和标准库的动态链接文件（主要是.a 文件和.so 文件）。而 JNI 开发只是在 App 工程下编写 C/C++代码，代码中包含 NDK 提供的头文件，build.gradle 或 mk 文件依据编译规则把标准库链接进去，编译通过后形成最终的 so 动态库文件，这样才能在 App 中通过 Java 代码调用 JNI 接口。

下面是 JNI 开发的具体步骤。

01 确保 NDK 环境搭建完成，并且本模块已经添加了对 NDK 的支持。

02 在要调用 JNI 接口的 Activity 代码中添加 JNI 接口定义，并在初始化时加载 JNI 动态库，具体代码举例如下：

```
public native String cpuFromJNI(int i1, float f1, double d1, boolean b1);
public native String unimplementedCpuFromJNI(int i1, float f1, double d1, boolean b1);
static {
    System.loadLibrary("jni_mix");
}
```

03 转到工程的 jni 目录下，在 h 文件、c 文件、cpp 文件中编写 C/C++代码。注意 C 代码中对接口名称的命名规则是“Java_包名_Activity 类名_函数名”。其中，包名中的点号要替换为下划线。下面是 C 代码对接口名称命名的代码：

```
jstring Java_com_example_mixture_JniCpuActivity_cpuFromJNI( JNIEnv* env, jobject thiz, jint i1, jfloat f1, jdouble d1, jboolean b1 )
```

04 在 build.gradle 中编写本模块的 NDK 编译规则，或另外在 jni 目录创建一个 mk 文件单独定义编译规则（如果在 build.gradle 中启用了 externalNativeBuild 节点）。

05 编译 JNI 代码，并把编译生成的 so 库复制到 jniLibs 目录，再重新运行 App。

以上开发步骤尚有 3 处需要补充描述，分别是数据类型转换、编译规则定义以及开发注意事项，详细说明如下：

1. 数据类型转换

JNI 作为 Java 与 C/C++之间的联系桥梁，需要对基本数据类型进行转换，基本数据类型的转换关系见表 14-2。

表14-2 基本数据类型的转换关系

数据类型名称	Java 的数据类型	JNI 的数据类型	C/C++的数据类型
整型	int	jint	int
浮点数	float	jfloat	float
双精度	double	jdouble	double

（续表）

数据类型名称	Java 的数据类型	JNI 的数据类型	C/C++的数据类型
布尔型	boolean	jboolean	unsigned char
字符串	String	jstring	const char*

其中，整型、浮点数、双精度 3 种数据类型可以由 C/C++直接使用，而布尔型和字符串需要处理后才能由 C/C++使用，具体的处理规则如下：

（1）处理布尔类型时，Java 的 false 对应 C/C++的 0，Java 的 true 对应 C/C++的 1。

（2）处理字符串类型时，JNI 使用 env→GetStringUTFChars 方法将 jstring 类型转为 const char*类型，使用 env→NewStringUTF 方法将 const char*类型转为 jstring 类型。

2. 编译规则定义

Android Studio 编译 C/C++代码有两种方式，分别是在 build.gradle 中编写编译规则和另外书写 Android.mk 定义编译规则。两种编译方式的规则定义大同小异，主要是规则名称的差异，编译规则名称的对应关系见表 14-3。

表14-3 编译规则名称的对应关系

build.gradle 的规则名称	Android.mk 的规则名称	说明	常用值
moduleName	LOCAL_MODULE	so 库文件的名称	
无	LOCAL_SRC_FILES	需要编译的源文件	
cFlags	LOCAL_CPPFLAGS	C++的编译标志	-fexceptions（支持 try..catch..）
ldLibs	LOCAL_LDLIBS	需要链接的库，多个库用逗号分隔	log（支持打印日志）
stl	APP_STL	stl 库的集成方式	stlport_static（表示使用 STLport 作为静态库）
abiFilters	APP_ABI	目标库的指令集，多个指令集用逗号分隔	armeabi-v7a（表示高级的 ARM）

3. 开发注意事项

由于 JNI 接口使用另一种语言开发，因此要注意克服 Java 单独编码或 C/C++单独编码的固定思维，需要注意以下事项：

（1）C/C++代码中的变量都要初始化，因为在真机上如果不初始化，值就不可预知，进而影响业务逻辑处理。

（2）由于 JNI 的接口名称包含包名、类名和函数名，因此务必保证该名称所表达的路径与 Java 代码完全一致，才能由 Java 代码正常调用 JNI 接口。

（3）JNI 中操作 socket 要设置上网权限，否则 socket 函数总是返回-1；以此类推，JNI 中操作 SD 卡文件存取也要设置 SD 卡权限。

接下来通过一个获取 CPU 指令集的例子演示一下 JNI 开发的完整流程和基本数据类型的

转换。下面是 JNI 代码文件 get_cpu.cpp 的源代码：

```
#include <jni.h>
#include <string.h>
#include <stdio.h>

extern "C"

jstring Java_com_example_mixture_JniCpuActivity_cpuFromJNI( JNIEnv* env, jobject thiz, jint i1, jfloat f1,
jdouble d1, jboolean b1 ) {
#if defined(__arm__)
  #if defined(__ARM_ARCH_7A__)
    #if defined(__ARM_NEON__)
      #if defined(__ARM_PCS_VFP)
        #define ABI "armeabi-v7a/NEON (hard-float)"
      #else
        #define ABI "armeabi-v7a/NEON"
      #endif
    #else
      #if defined(__ARM_PCS_VFP)
        #define ABI "armeabi-v7a (hard-float)"
      #else
        #define ABI "armeabi-v7a"
      #endif
    #endif
  #else
   #define ABI "armeabi"
  #endif
#elif defined(__i386__)
   #define ABI "x86"
#elif defined(__x86_64__)
   #define ABI "x86_64"
#elif defined(__mips64)  /* mips64el-* toolchain defines __mips__ too */
   #define ABI "mips64"
#elif defined(__mips__)
   #define ABI "mips"
#elif defined(__aarch64__)
   #define ABI "arm64-v8a"
#else
   #define ABI "unknown"
#endif
     char desc[200] = {0};
     sprintf(desc, "%d %f %lf %u \nHello from JNI !   Compiled with %s.", i1, f1, d1, b1, ABI);
```

```
        return env->NewStringUTF(desc);
    }
```

下面是活动页面的 Java 代码，先从 Build 类获取当前的指令集，再调用 JNI 接口获取 C++ 代码得到的指令集：

```
public class JniCpuActivity extends AppCompatActivity implements OnClickListener {
    private TextView tv_cpu_build;
    private TextView tv_cpu_jni;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_jni_cpu);
        tv_cpu_build = (TextView) findViewById(R.id.tv_cpu_build);
        tv_cpu_jni = (TextView) findViewById(R.id.tv_cpu_jni);
        findViewById(R.id.btn_cpu).setOnClickListener(this);
        tv_cpu_build.setText("Build 类获得的 CPU 指令集为"+Build.CPU_ABI);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_cpu) {
            String desc = cpuFromJNI(1, 0.5f, 99.9, true);
            tv_cpu_jni.setText(desc);
        }
    }

    public native String cpuFromJNI(int i1, float f1, double d1, boolean b1);
    public native String unimplementedCpuFromJNI(int i1, float f1, double d1, boolean b1);
    static {
        System.loadLibrary("jni_mix");
    }
}
```

JNI 接口获取指令集的结果如图 14-15 和图 14-16 所示。图 14-15 所示为模拟器上的运行结果截图。图 14-16 所示为真机上的运行结果截图。

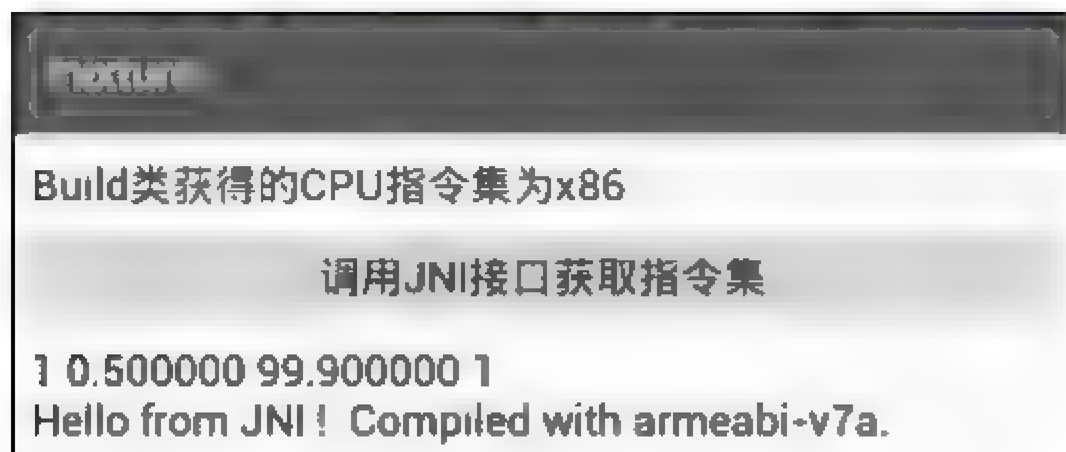


图 14-15　模拟器获得的指令集

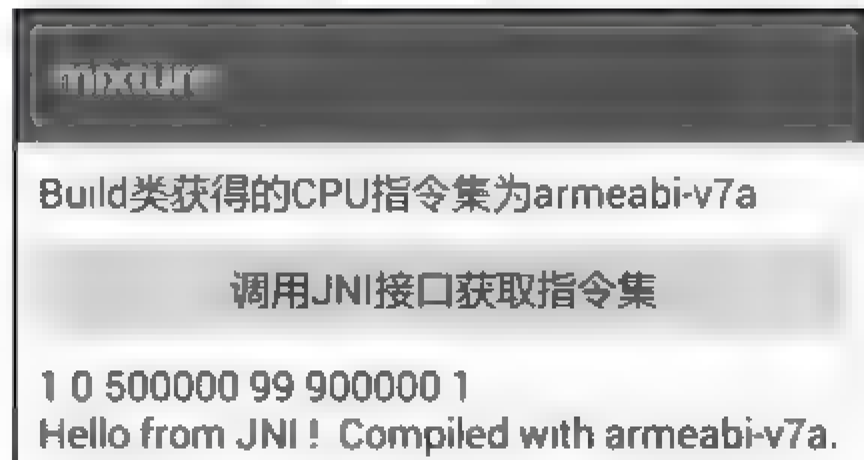


图 14-16　真机获得的指令集

14.2.3 JNI 实现加解密

实际开发中，JNI 主要应用于如下业务场景：

1. 对关键业务数据进行加解密

虽然 Java 提供了常用的加解密方法，但是 Java 代码容易遭到破解，而 so 库到目前为止是不可破解的，所以使用 JNI 进行加解密无疑更加安全。

2. 底层的网络操作与设备操作

Java 作为一门高级语言，与硬件和网络操作的隔阂比 C/C++大，不像 C/C++那样容易驾驭底层操作。

3. 对运行效率要求较高的场合

同样的操作，C/C++的执行效率比 Java 高得多，iOS 基于 C/C++的变种 ObjectC，而 Android 基于 Java，所以 iOS 的流畅性强于 Android。Android 上的 SQLite 使用 Java 实现，因此性能存在瓶颈。现在移动端兴起了第三方的数据库 Realm，性能优异渐有取代 SQLite 之势，而 Realm 的底层是用 C/C++实现的。

4. 跨平台的应用移植

移动设备的操作系统不是 Android 就是 iOS，现在企业开发 App 一般都要做两条产品线，一条做 Android，另一条做 iOS，同样的功能需要两边分别实现，费时费力。如果部分业务功能采用 C/C++实现，那么不但 Android 可以通过 JNI 调用，而且 iOS 能直接编译运行，一份代码可同时被两个平台复用，省时省力。

接下来我们尝试使用 JNI 完成加解密操作。C/C++的加解密算法代码不少，本书采用的是 C++的 AES 算法开源代码，主要的改造工作是给 C++源代码配上 JNI 接口。

下面是 JNI 接口的 AES 加密代码：

```
#include <jni.h>
#include <string.h>
#include <stdio.h>
#include "aes.h"
#include <android/log.h>
// log 标签
#define TAG "MyMsg"
// 定义 info 信息
#define LOGI(...) __android_log_print(ANDROID_LOG_INFO,TAG,__VA_ARGS__)

extern "C"

jstring Java_com_example_mixture_JniSecretActivity_encryptFromJNI( JNIEnv* env, jobject thiz, jstring
raw, jstring key) {
```

```
        const char* str_raw;
        const char* str_key;
        str_raw = env->GetStringUTFChars(raw, 0);
        str_key = env->GetStringUTFChars(key, 0);
        LOGI("str_raw=%s, str_key=%s ", str_raw, str_key);
        char encrypt[1024] = {0};
        AES aes_en((unsigned char*)str_key);
        aes_en.Cipher((char*)str_raw, encrypt);
        LOGI("encrypt=%s", encrypt);
        return env->NewStringUTF(encrypt);
    }
```

下面是 JNI 接口的 AES 解密代码：

```
    #include <jni.h>
    #include <string.h>
    #include <stdio.h>
    #include "aes.h"
    #include <android/log.h>
    // log 标签
    #define TAG "MyMsg"
    // 定义 info 信息
    #define LOGI(...) __android_log_print(ANDROID_LOG_INFO,TAG,__VA_ARGS__)

    extern "C"

    jstring Java_com_example_mixture_JniSecretActivity_decryptFromJNI( JNIEnv* env, jobject thiz, jstring des,
jstring key) {
        const char* str_des;
        const char* str_key;
        str_des = env->GetStringUTFChars(des, 0);
        str_key = env->GetStringUTFChars(key, 0);
        LOGI("str_des=%s, str_key=%s ", str_des, str_key);
        char decrypt[1024] = {0};
        AES aes_de((unsigned char*)str_key);
        aes_de.InvCipher((char*)str_des, decrypt);
        LOGI("decrypt=%s", decrypt);
        return env->NewStringUTF(decrypt);
    }
```

下面是活动页面的 Java 代码，通过界面对输入数据进行加解密操作：

```
    public class JniSecretActivity extends AppCompatActivity implements OnClickListener {
        private EditText et_origin;
        private EditText et_encrypt;
        private TextView tv_cpu_build;
```

```
    private TextView tv_decrypt;
    private String mKey = "123456789abcdef"; //该算法要求密钥值长度为 16 位

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_jni_secret);
        et_origin = (EditText) findViewById(R.id.et_origin);
        et_encrypt = (EditText) findViewById(R.id.et_encrypt);
        tv_decrypt = (TextView) findViewById(R.id.tv_decrypt);
        findViewById(R.id.btn_encrypt).setOnClickListener(this);
        findViewById(R.id.btn_decrypt).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_encrypt) {
            String des = encryptFromJNI(et_origin.getText().toString(), mKey);
            et_encrypt.setText(des);
        } else if (v.getId() == R.id.btn_decrypt) {
            String raw = decryptFromJNI(et_encrypt.getText().toString(), mKey);
            tv_decrypt.setText(raw);
        }
    }

    public native String encryptFromJNI(String raw, String key);
    public native String unimplementedEncryptFromJNI(String raw, String key);
    public native String decryptFromJNI(String des, String key);
    public native String unimplementedEecryptFromJNI(String des, String key);
    static {
        System.loadLibrary("jni_mix");
    }
}
```

JNI 实现加解密的效果如图 14-17 和图 14-18 所示。图 14-17 所示为输入原始字符串并调用 JNI 接口进行加密的结果界面。图 14-18 所示为对加密串进行 JNI 解密操作的结果界面。



图 14-17　JNI 的加密结果



图 14-18　JNI 的解密结果

14.3 局域网共享

本节介绍融合技术的一个重要方向——局域网共享，包括文件在内的手机资源都有可能利用局域网技术分享给其他设备。本节首先说明如何使用无线网络管理器获取当前的WIFI信息，然后详细阐述蓝牙技术的4个工具组件，以及如何利用蓝牙技术实现两台设备之间的消息传递。

14.3.1 无线网络管理器 WifiManager

第10章提到，App若想访问外网资源，得先判断网络连接是否可用。当时检测连接的工具采用了连接管理器ConnectivityManager，上网方式主要有两种，即数据连接和WIFI。不过ConnectivityManager只能笼统的判断能否上网，并不能获知WIFI连接的详细信息。当前网络类型是WIFI时，要想得知WIFI上网的具体信息，需另外通过无线网络管理器WifiManager获取。

WifiManager的对象从系统服务Context.WIFI_SERVICE中获取。下面是WifiManager的常用方法。

- isWifiEnabled：判断WLAN功能是否开启。
- setWifiEnabled：开启或关闭WLAN功能。
- getWifiState：获取当前的WIFI连接状态。WIFI连接状态的取值说明见表14-4。

表14-4 WIFI连接状态的取值说明

WifiManager 类的连接状态	说明
WIFI_STATE_DISABLED	已断开 WIFI
WIFI_STATE_DISABLING	正在断开 WIFI
WIFI_STATE_ENABLED	已连上 WIFI
WIFI_STATE_ENABLING	正在连接 WIFI
WIFI_STATE_UNKNOWN	连接状态未知

- getConnectionInfo：获取当前WIFI的连接信息。该方法返回一个WifiInfo对象，通过该对象的各个方法可获得更具体的WIFI设备信息。下面是信息获取方法说明。
  - getSSID：WIFI路由器MAC。
  - getRssi：WIFI信号强度。
  - getLinkSpeed：连接速率。
  - getNetworkId：WIFI的网络编号。
  - getIpAddress：手机的IP地址。整型数，需转换为常见的IPv4地址。
  - getMacAddress：手机的MAC地址。
- startScan：开始扫描周围的WIFI信息。



- getScanResults：获取 WIFI 的扫描结果。
- calculateSignalLevel：根据信号强度计算信号等级。
- getConfiguredNetworks：获取已配置的网络信息。
- addNetwork：添加指定的 WIFI 连接。
- enableNetwork：启用指定的 WIFI 连接。第二个参数表示是否同时禁用其他 WIFI。
- disableNetwork：禁用指定的 WIFI 连接。
- disconnect：断开当前的 WIFI 连接。

查看 WIFI 连接信息的实现代码很简单，读者可自行实践。WIFI 信息的查看效果如图 14-19 所示，主要包括 WIFI 路由器的相关信息、手机在该 WIFI 环境下分配到的 IP 地址和 MAC 地址。



图 14-19 真机获取到的 WIFI 信息

14.3.2 蓝牙 BlueTooth

无论是 WIFI 还是 4G 网络，建立网络连接后都是访问互联网资源，并不能直接访问局域网资源。比如两个人在一起，A 要把手机上的视频传给 B，通常情况是打开手机 QQ，通过 QQ 传送文件给对方。不过上传视频很耗流量，如果现场没有可用的 WIFI，手机的数据流量又不足，就只能干瞪眼了。为解决这种邻近传输文件的问题，蓝牙技术应运而生。蓝牙技术是一种无线技术标准，可实现设备之间的短距离数据交换。

Android 为蓝牙技术提供了 4 个工具类，分别是蓝牙适配器 BuletoothAdapter、蓝牙设备 BluetoothDevice、蓝牙服务端套接字 BluetoothServerSocket 和蓝牙客户端套接字 BluetoothSocket。

1. 蓝牙适配器 BuletoothAdapter

BuletoothAdapter 的作用其实跟其他的***Manager 差不多，可以把它当作蓝牙管理器。下面是 BuletoothAdapter 的常用方法说明。

- getDefaultAdapter：静态方法，获取默认的蓝牙适配器对象。
- enable：打开蓝牙功能。该方法在打开蓝牙时不会弹出提示，所以一般不这么调用。更常见的做法是弹出对话框，提示用户是否允许外部发现本设备。因为只有让外部设备发现本设备，才能够进行后续配对与连接操作。弹窗提示用户打开蓝牙功能的代码如下：

```
Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
startActivityForResult(intent, 1);
```

- disable：关闭蓝牙功能。
- isEnabled：判断蓝牙功能是否打开。已打开就返回 true，否则返回 false。
- startDiscovery：开始搜索周围的蓝牙设备。搜索结果通过广播返回。
- cancelDiscovery：取消搜索操作。
- isDiscovering：判断当前是否正在搜索设备。
- getBondedDevices：获取已绑定的设备列表。该方法返回的是已绑定设备的历史记录，而

非当前能够连接的设备。

- setName：设置本机的蓝牙名称。
- getName：获取本机的蓝牙名称。
- getAddress：获取本机的蓝牙地址。
- getRemoteDevice：根据蓝牙地址获取远程的蓝牙设备。
- getState：获取本地蓝牙适配器的状态。值为 BluetoothAdapter.STATE_ON 表示蓝牙可用。
- listenUsingRfcommWithServiceRecord：根据名称和 UUID 创建并返回 BluetoothServerSocket。
- listenUsingRfcommOn：根据渠道编号创建并返回 BluetoothServerSocket。

2. 蓝牙设备 BluetoothDevice

BluetoothDevice 用于指代某个蓝牙设备，通常表示对方设备。BuletoothAdapter 管理的是本机的蓝牙设备。下面是 BluetoothDevice 的常用方法说明。

- getName：获得该设备的名称。
- getAddress：获得该设备的地址。
- getBondState：获得该设备的绑定状态。蓝牙设备绑定状态的取值说明见表 14-5。

表14-5　蓝牙设备绑定状态的取值说明

BluetoothDevice 类的绑定状态	说明
BOND_NONE	未绑定（未配对）
BOND_BONDING	正在绑定（正在配对）
BOND_BONDING	已绑定（已配对）

- createBond：创建配对请求。配对结果通过广播返回。
- createRfcommSocketToServiceRecord：根据 UUID 创建并返回一个 BluetoothSocket。
- createRfcommSocket：根据渠道编号创建并返回一个 BluetoothSocket。

3. 蓝牙服务端套接字 BluetoothServerSocket

BluetoothServerSocket 是服务端的 Socket，用来接收客户端的 socket 连接请求。下面是常用方法说明。

- accept：监听外部的蓝牙连接请求。一旦有请求接入，就返回一个 BluetoothSocket 对象。
- close：关闭服务端的蓝牙监听。

4. 蓝牙客户端套接字 BluetoothSocket

BluetoothSocket 是客户端的 Socket，用于与对方设备进行数据通信。下面是常用方法说明。

- connect：建立蓝牙的 socket 连接。
- close：关闭蓝牙的 socket 连接。
- getInptuStream：获取 socket 连接的输入流对象。
- getOutputStream：获取 socket 连接的输出流对象。

- getRemoteDevice：获取远程设备信息，即与本设备建立 socket 连接的远程蓝牙设备。

上述工具的介绍有点枯燥乏味，接下来演示使用蓝牙建立连接、发送消息的完整流程，有了直观印象才能进一步理解蓝牙开发的具体过程。完整流程主要分为以下 5 个步骤：

1. 开启蓝牙功能

准备两部手机，各自安装蓝牙演示 App。首先打开演示 App 的蓝牙页面，一开始两部手机的蓝牙功能均为关闭，初始状态的页面效果如图 14-20 所示。



图 14-20 蓝牙 DEMO 工程的初始页面

分别点击两部手机左上角的开关按钮，准备开启手机的蓝牙功能。两部手机都弹出一个确认对话框，提示用户是否允许其他设备检测到本手机。此时，A 手机的授权弹窗页面如图 14-21 所示，B 手机的授权弹窗页面如图 14-22 所示。

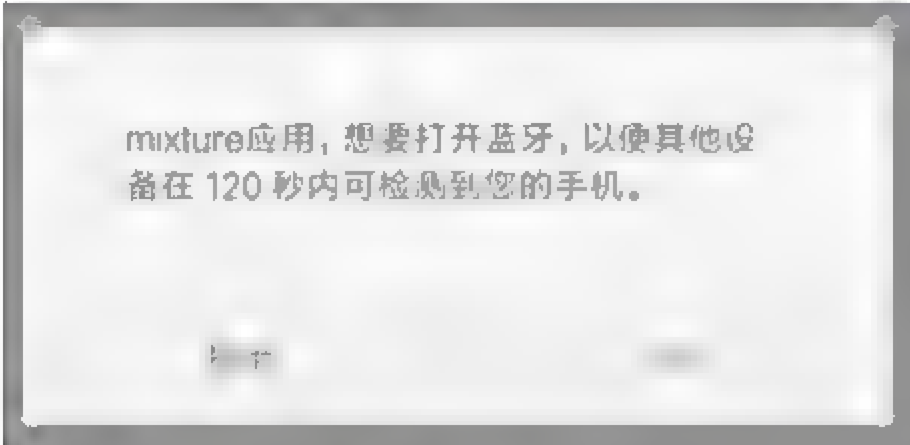


图 14-21 A 手机的授权弹窗

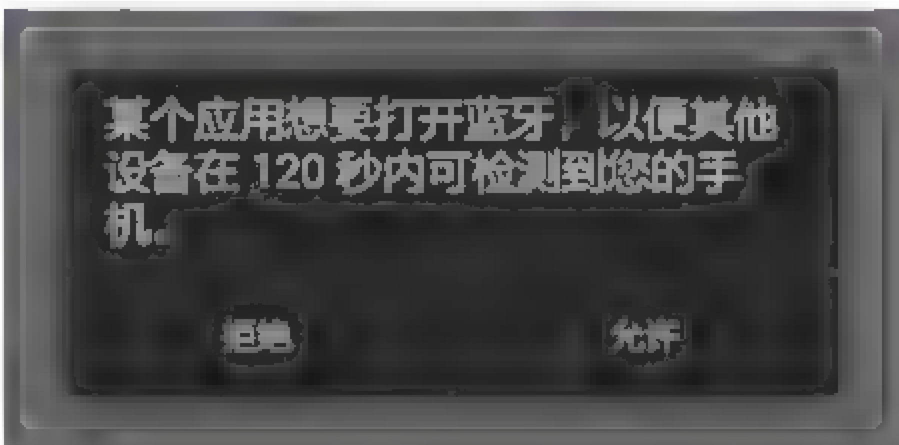


图 14-22 B 手机的授权弹窗

当然，都要点击“允许”按钮确认开启蓝牙功能。稍等一会儿，两部手机分别检测到了对方设备的存在，把对方设备显示在页面上，状态为“未绑定”。此时 A 手机的页面信息如图 14-23 所示，B 手机的页面信息如图 14-24 所示。



图 14-23 A 手机发现对方



图 14-24 B 手机发现对方

2. 确认配对并完成绑定

在任意一部手机上点击对方设备的记录，表示发起配对请求。两部手机都弹出一个确认对话框，提示用户是否将本机与对方设备进行配对。此时，A 手机的配对弹窗页面如图 14-25 所示，B 手机的配对弹窗页面如图 14-26 所示。

两边分别点击“配对”按钮，确认与对方进行配对操作。配对完成后，蓝牙页面上将对方设备的状态改为“已绑定”。此时，A 手机的页面信息如图 14-27 所示，B 手机的页面信息如图 14-28 所示。



图 14-25　A 手机的配对弹窗

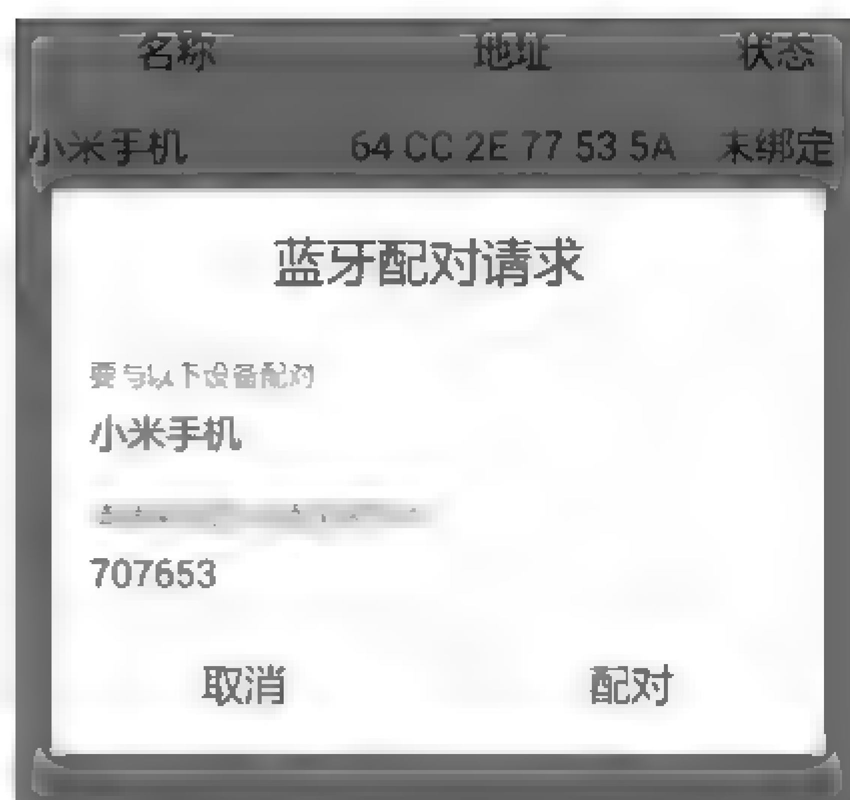


图 14-26　B 手机的配对弹窗



图 14-27　A 手机完成配对



图 14-28　B 手机完成配对

3. 建立蓝牙连接

在任意一部手机上点击已绑定的设备记录，表示发起连接请求。具体地说，首先是客户端的 BluetoothSocket 调用 connect 方法，然后服务端 BluetoothServerSocket 的 accept 方法接收连接请求，于是双方成功建立连接。有的手机可能会弹窗提示"应用***想与***设备进行通信"，点击弹窗的"确定"按钮即可放行。建立连接后，设备记录右边的状态值改为"已连接"。此时，A 手机的页面信息如图 14-29 所示，B 手机的页面信息如图 14-30 所示。



图 14-29　A 手机与对方建立连接



图 14-30　B 手机与对方建立连接

4. 通过蓝牙发送消息

在 A 手机上点击已连接的设备记录，表示想要发送消息。于是 A 手机弹出文字输入对话框，提示用户输入待发送的消息文本，文字输入框效果如图 14-31 所示。点击"确定"按钮发送消息，B 手机接收到 A 手机发来的消息，就把该消息文本通过弹窗显示出来，B 手机的消息弹窗效果如图 14-32 所示。



图 14-31　A 手机准备向对方发送消息



图 14-32　B 手机收到对方发来的消息

至此，一个完整的蓝牙应用过程就全部呈现出来了。上面的流程仅实现了简单的字符串传输，真实场景更需要文件传输。当然，使用输入输出流操作文件也不是什么难事。不过蓝牙开发需要两部手机一起操作，确实有点复杂，中间还有不少坑，真是一言难尽。读者不如自行阅读 demo 源码并动手实践，这样才能收获真知灼见。

注意使用蓝牙功能需要为 App 赋权，即在 AndroidManifest.xml 中增加如下权限定义：

```
<!-- 蓝牙 -->
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.BLUETOOTH" />
```

下面是建立蓝牙连接并发送消息的页面代码，服务端与客户端通用，更多任务代码见本书的下载资源：

```
public class BluetoothActivity extends AppCompatActivity implements
        OnClickListener, OnItemClickListener, OnCheckedChangeListener,
        BlueConnectListener, InputCallbacks, BlueAcceptListener {
    private static final String TAG = "BluetoothActivity";
    private CheckBox ck_bluetooth;
    private TextView tv_discovery;
    private ListView lv_bluetooth;
    private BluetoothAdapter mBluetooth;
    private ArrayList<BlueDevice> mDeviceList = new ArrayList<BlueDevice>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_bluetooth);
        ck_bluetooth = (CheckBox) findViewById(R.id.ck_bluetooth);
        tv_discovery = (TextView) findViewById(R.id.tv_discovery);
```

```
        lv_bluetooth = (ListView) findViewById(R.id.lv_bluetooth);
        if (BluetoothUtil.getBlueToothStatus(this) == true) {
            ck_bluetooth.setChecked(true);
        }
        ck_bluetooth.setOnCheckedChangeListener(this);
        tv_discovery.setOnClickListener(this);
        mBluetooth = BluetoothAdapter.getDefaultAdapter();
        if (mBluetooth == null) {
            Toast.makeText(this, "本机未找到蓝牙功能", Toast.LENGTH_SHORT).show();
            finish();
        }
    }

    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (buttonView.getId() == R.id.ck_bluetooth) {
            if (isChecked == true) {
                beginDiscovery();
                Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
                startActivityForResult(intent, 1);
                // 下面这行代码为服务端需要，客户端不需要
                mHandler.postDelayed(mAccept, 1000);
            } else {
                cancelDiscovery();
                BluetoothUtil.setBlueToothStatus(this, false);
                mDeviceList.clear();
                BlueListAdapter adapter = new BlueListAdapter(this, mDeviceList);
                lv_bluetooth.setAdapter(adapter);
            }
        }
    }

    private Runnable mAccept = new Runnable() {
        @Override
        public void run() {
            if (mBluetooth.getState() == BluetoothAdapter.STATE_ON) {
                BlueAcceptTask acceptTask = new BlueAcceptTask(true);
                acceptTask.setBlueAcceptListener(BluetoothActivity.this);
                acceptTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
            } else {
                mHandler.postDelayed(this, 1000);
            }
        }
    };
```

```
        @Override
        public void onClick(View v) {
            if (v.getId() == R.id.tv_discovery) {
                beginDiscovery();
            }
        }

        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
            super.onActivityResult(requestCode, resultCode, intent);
            if (requestCode == 1) {
                if (resultCode == RESULT_OK) {
                    Toast.makeText(this, "允许本地蓝牙被附近其他蓝牙设备发现",
Toast.LENGTH_SHORT).show();
                } else if (resultCode == RESULT_CANCELED) {
                    Toast.makeText(this, "不允许蓝牙被附近其他蓝牙设备发现",
Toast.LENGTH_SHORT).show();
                }
            }
        }

        private Runnable mRefresh = new Runnable() {
            @Override
            public void run() {
                beginDiscovery();
                mHandler.postDelayed(this, 2000);
            }
        };

        private void beginDiscovery() {
            if (mBluetooth.isDiscovering() != true) {
                mDeviceList.clear();
                BlueListAdapter adapter = new BlueListAdapter(BluetoothActivity.this, mDeviceList);
                lv_bluetooth.setAdapter(adapter);
                tv_discovery.setText("正在搜索蓝牙设备");
                mBluetooth.startDiscovery();
            }
        }

        private void cancelDiscovery() {
            mHandler.removeCallbacks(mRefresh);
            tv_discovery.setText("取消搜索蓝牙设备");
            if (mBluetooth.isDiscovering() == true) {
                mBluetooth.cancelDiscovery();
            }
```

```
    }

    @Override
    protected void onStart() {
        super.onStart();
        mHandler.postDelayed(mRefresh, 50);
        blueReceiver = new BluetoothReceiver();
        //需要过滤多个动作，则调用 IntentFilter 对象的 addAction 添加新动作
        IntentFilter foundFilter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        foundFilter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        foundFilter.addAction(BluetoothDevice.ACTION_BOND_STATE_CHANGED);
        registerReceiver(blueReceiver, foundFilter);
    }

    @Override
    protected void onStop() {
        super.onStop();
        cancelDiscovery();
        unregisterReceiver(blueReceiver);
    }

    private BluetoothReceiver blueReceiver;
    private class BluetoothReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            Log.d(TAG, "onReceive action="+action);
            // 获得已经搜索到的蓝牙设备
            if (action.equals(BluetoothDevice.ACTION_FOUND)) {
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                BlueDevice item = new BlueDevice(device.getName(), device.getAddress(),
device.getBondState()-10);
                mDeviceList.add(item);
                BlueListAdapter adapter = new BlueListAdapter(BluetoothActivity.this, mDeviceList);
                lv_bluetooth.setAdapter(adapter);
                lv_bluetooth.setOnItemClickListener(BluetoothActivity.this);
            } else if (action.equals(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)) {
                mHandler.removeCallbacks(mRefresh);
                tv_discovery.setText("蓝牙设备搜索完成");
            } else if (action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)) {
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if (device.getBondState() == BluetoothDevice.BOND_BONDING) {
                    tv_discovery.setText("正在配对"+device.getName());
                } else if (device.getBondState() == BluetoothDevice.BOND_BONDED) {
```

```
                    tv_discovery.setText("完成配对"+device.getName());
                    mHandler.postDelayed(mRefresh, 50);
                } else if (device.getBondState() == BluetoothDevice.BOND_NONE) {
                    tv_discovery.setText("取消配对"+device.getName());
                }
            }
        }
    }

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        cancelDiscovery();
        BlueDevice item = mDeviceList.get(position);
        BluetoothDevice device = mBluetooth.getRemoteDevice(item.address);
        try {
            if (device.getBondState() == BluetoothDevice.BOND_NONE) {
                Method createBondMethod = BluetoothDevice.class.getMethod("createBond");
                Log.d(TAG, "开始配对");
                Boolean result = (Boolean) createBondMethod.invoke(device);
            } else if (device.getBondState() == BluetoothDevice.BOND_BONDED &&
                    item.state != BlueListAdapter.CONNECTED) {
                tv_discovery.setText("开始连接");
                BlueConnectTask connectTask = new BlueConnectTask(item.address);
                connectTask.setBlueConnectListener(this);
                connectTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR, device);
            } else if (device.getBondState() == BluetoothDevice.BOND_BONDED &&
                    item.state == BlueListAdapter.CONNECTED) {
                tv_discovery.setText("正在发送消息");
                InputDialogFragment dialog = InputDialogFragment.newInstance(
                        "", 0, "请输入要发送的消息");
                String fragTag = getResources().getString(R.string.app_name);
                dialog.show(getFragmentManager(), fragTag);
            }
        } catch (Exception e) {
            e.printStackTrace();
            tv_discovery.setText("配对异常 ："+e.getMessage());
        }
    }

    //向对方发送消息
    @Override
    public void onInput(String title, String message, int type) {
        Log.d(TAG, "onInput message="+message);
        BluetoothUtil.writeOutputStream(mBlueSocket, message);
```

```
}

private BluetoothSocket mBlueSocket;
//客户端主动连接
@Override
public void onBlueConnect(String address, BluetoothSocket socket) {
        mBlueSocket = socket;
        tv_discovery.setText("连接成功");
        refreshAddress(address);
}

//刷新已连接的状态
private void refreshAddress(String address) {
        for (int i=0; i<mDeviceList.size(); i++) {
                BlueDevice item = mDeviceList.get(i);
                if (item.address.equals(address) == true) {
                        item.state = BlueListAdapter.CONNECTED;
                        mDeviceList.set(i, item);
                }
        }
        BlueListAdapter adapter = new BlueListAdapter(this, mDeviceList);
        lv_bluetooth.setAdapter(adapter);
}

//服务端侦听到连接
@Override
public void onBlueAccept(BluetoothSocket socket) {
        Log.d(TAG, "onBlueAccept socket is "+(socket==null?"null":"not null"));
        if (socket != null) {
                mBlueSocket = socket;
                BluetoothDevice device = mBlueSocket.getRemoteDevice();
                refreshAddress(device.getAddress());
                BlueReceiveTask receive = new BlueReceiveTask(mBlueSocket, mHandler);
                receive.start();
        }
}

//收到对方发来的消息
private Handler mHandler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
                if (msg.what == 0) {
                        byte[] readBuf = (byte[]) msg.obj;
                        String readMsg = new String(readBuf, 0, msg.arg1);
                        Log.d(TAG, "handleMessage readMessage="+ readMsg);
```

```
                    AlertDialog.Builder builder = new AlertDialog.Builder(BluetoothActivity.this);
                    builder.setTitle("我收到消息啦").setMessage(readMsg).setPositiveButton("确定", null);
                    builder.create().show();
                }
            }
        };

        @Override
        protected void onDestroy() {
            super.onDestroy();
            if (mBlueSocket != null) {
                try {
                    mBlueSocket.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

14.4 实战项目：共享经济弄潮儿——WIFI 共享器

互联网之所以成为新经济，很重要的一个原因是深入贯彻了分享的精髓。从你有我无，到人人共享，这是人类历史上的一大进步。本章介绍的融合相关技术从一起显示网页到一起运行代码，再到一起分享文件，其实都渗透着共享的思想。本章结尾进一步在用户手机之间共享网络，即共享流量。具体地说，就是一个手机开启 WIFI 热点，其他设备均可接入该 WIFI 上网冲浪，这就是本章的实战项目——“WIFI 共享器”。

14.4.1 设计思路

大家应该都有外出游玩的经历，因为室外 WIFI 信号很弱，要么干脆找不到公共 WIFI 信号，所以在外玩耍往往很消耗手机流量。有的朋友用完了流量，有的朋友还剩不少流量，能不能把剩余的流量给别人使用呢？当然可以。现在不少手机都自带个人热点/WLAN 热点/WIFI 热点之类的功能，开启该功能即可将手机变为一台无线路由器，其他设备连接该手机的热点 WIFI 后上网就不会耗费自身流量，而是使用开启热点的手机的数据流量。

手机的 WIFI 热点功能一般集成在系统设置中，页面很简单，功能也相对简单。现在我们给热点做一下功能增强，实现名副其实的 WIFI 共享器，页面效果如图 14-33 所示。

光看这个页面，读者可能不能很快明白采用了哪些 App 技术，且待笔者细细数来，看看这个简单的页面究竟蕴含哪些江湖绝技。



（1）无线网络管理器WifiManager：这是比较明白的，开关热点都要由WifiManager操作。

（2）系统文件读取：在系统文件/proc/net/arp 中，可找到已连接设备列表的 IP 和 MAC 地址。

（3）网络地址 InetAddress：判断某个设备能否连上，用到了 InetAddress 的 isReachable 方法。

（4）异步任务 AsyncTask：涉及网络操作都要把处理逻辑放在子线程中处理。

（5）资产管理器 AssetManager：用于导入 MAC 地址与设备厂商的对应关系表。因为联网设备的 MAC 由国际电子协会 IEEE 统一分配，未经认证和授权的厂家无权生产，MAC 地址的前 6 位代表手机和电脑厂商，所以可通过 MAC 地址查询对应的厂商名称。MAC 与厂商的对应关系可从 http://standards.ieee.org/regauth/ oui/oui.txt 查询。



图 14-33　WIFI 共享器的效果图

（6）数据库 SQLite：MAC 与厂商关系表要导入 SQLite 数据库中，方便后续查询操作。

（7）异步服务 IntentService：从 assets 目录导入 MAC 与厂商关系表，由于比较耗时，因此为了避免页面挂死，必须开启后台服务执行导入操作，而且开启子线程的异步服务。

（8）套接字 Socket：使用 socket 技术通过 NetBIOS 协议获取网络上的计算机名称。

（9）JNI 开发：C 语言完成 NetBIOS 协议的信息获取需要实现 JNI 接口供 Java 代码调用。

真是不数不知道，原来简简单单的页面背后竟隐藏了这么多不为人知的高招，所以不要小看 App 开发，做好一项功能往往需要联合使用多种技术。

14.4.2　小知识：NetBIOS 协议

NetBIOS 协议是一种局域网上的应用程序编程接口，为程序提供了请求低级服务的统一命令集，允许程序和网络会话。在 Windows 操作系统中，安装 TCP/IP 协议后会自动安装 NetBIOS。也就是说，Windows 平台自带 NetBIOS 服务。

NetBIOS 提供的名字包括计算机名称、工作组名和域名。从程序角度来看，只要一个 IP 地址用的是 Windows 操作系统，通过 NetBIOS 协议即可获得该 IP 的计算机名和 MAC 地址，为开发者获知对方的设备信息提供了便利。

其实，通过 Java 代码就能根据 IP 地址获取对方的计算机名，示例代码见下载资源本章源码包里的 GetClientName.java，具体的执行结果如图 14-34 所示。

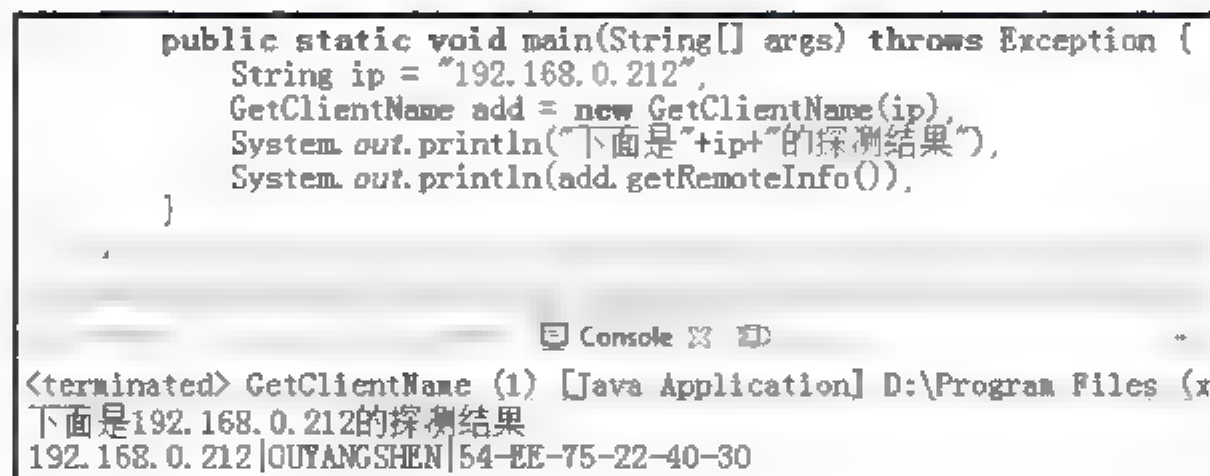


图 14-34　Java 代码实现 NetBIOS 协议

在底层操作 NetBIOS 怎么少得了威名赫赫的 C 语言呢？正好本章介绍了 JNI 开发，不妨使用 C 语言的代码实现计算机名的获取功能。下面是完整的 JNI 代码，读者可尝试将其集成到实战项目中：

```
#include <jni.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netdb.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/select.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define send_MAXSIZE 50
#define recv_MAXSIZE 1024
struct NETBIOSNS {
        unsigned short int tid;         //unsigned short int 占 2 字节
        unsigned short int flags;
        unsigned short int questions;
        unsigned short int answerRRS;
        unsigned short int authorityRRS;
        unsigned short int additionalRRS;
        unsigned char name[34];
        unsigned short int type;
        unsigned short int classe;
};
char *getNameFromIp(const char *ip);

extern "C"

jstring Java_com_example_mixture_WifiShareActivity_nameFromJNI( JNIEnv* env, jobject thiz, jstring ip) {
        const char* str_ip;
        str_ip = env->GetStringUTFChars(ip, 0);
        return env->NewStringUTF(getNameFromIp(str_ip));
}

char *getNameFromIp(const char *ip) {
        char str_info[1024] = {0};
        struct sockaddr_in toAddr;          //在 sendto 中使用的对方地址
```

```
        struct sockaddr_in fromAddr;        //在 recvfrom 中使用的对方主机地址
        char send_buff[send_MAXSIZE];
        char recv_buff[recv_MAXSIZE];
        memset(send_buff, 0, sizeof(send_buff));
        memset(recv_buff, 0, sizeof(recv_buff));
        int sockfd; //socket
        unsigned int udp_port = 137;
        int inetat;
        if ( (inetat = inet_aton(ip, &toAddr.sin_addr)) == 0) {
            sprintf(str_info, "[%s] is not a valid IP address\n", ip);
            return str_info;
        }
        if ( (sockfd = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)) < 0) {
            sprintf(str_info, "%s socket error sockfd=%d, inetat=%d\n", ip, sockfd, inetat);
            return str_info;
        }
        bzero((char*)&toAddr,sizeof(toAddr));
        toAddr.sin_family = AF_INET;
        toAddr.sin_addr.s_addr = inet_addr(ip);
        toAddr.sin_port = htons(udp_port);
        //构造 NetBIOS 结构包
        struct NETBIOSNS nbns;
        nbns.tid=0x0000;
        nbns.flags=0x0000;
        nbns.questions=0x0100;
        nbns.answerRRS=0x0000;
        nbns.authorityRRS=0x0000;
        nbns.additionalRRS=0x0000;
        nbns.name[0]=0x20;
        nbns.name[1]=0x43;
        nbns.name[2]=0x4b;
        int j=0;
        for (j=3;j<34;j++) {
            nbns.name[j]=0x41;
        }
        nbns.name[33]=0x00;
        nbns.type=0x2100;
        nbns.classe=0x0100;
        memcpy(send_buff, &nbns, sizeof(nbns));
        int send_num =0;
        send_num = sendto(sockfd, send_buff, sizeof(send_buff), 0, (struct sockaddr *)&toAddr,
sizeof(toAddr) );
        if (send_num != sizeof(send_buff)) {
```

```
            sprintf(str_info, "%s sendto() error sockfd=%d, send_num=%d, sizeof(send_buff)=%d\n", ip,
sockfd, send_num, sizeof(send_buff));
            shutdown(sockfd, 2);
            return str_info;
        }
        int recv_num = recvfrom(sockfd, recv_buff, sizeof(recv_buff), 0,  (struct sockaddr *)NULL,
(socklen_t*)NULL);
        if (recv_num < 56) {
            sprintf(str_info, "%s recvfrom() error sockfd=%d, recv_num=%d\n", ip, sockfd, recv_num);
            shutdown(sockfd, 2);
            return str_info;
        }
        //这里要初始化。因为发现 Linux 和模拟器都没问题，真机上该变量如果不初始化，值就不可预知
        unsigned short int NumberOfNames=0;
        memcpy(&NumberOfNames, recv_buff+56, 1);
        char str_name[1024] = {0};
        unsigned short int mac[6]={0};
        int i=0;
        for (i=0; i<NumberOfNames; i++) {
            char NetbiosName[16];
            memcpy(NetbiosName, recv_buff+57+i*18, 16);  //依次读取 NetBIOS name
            if (i == 0) {
                sprintf(str_name, "%s", NetbiosName);
            }
        }
        sprintf(str_info, "%s|%s|", ip, str_name);
        for (i=0; i<6; i++) {
            memcpy(&mac[i], recv_buff+57+NumberOfNames*18+i,1);
            sprintf(str_info, "%s%02X", str_info, mac[i]);
            if (i != 5) {
                sprintf(str_info, "%s-", str_info);
            }
        }
        return str_info;
    }
```

14.4.3 代码示例

编码与测试方面需要注意以下几点：

（1）MAC 地址与设备厂商的对应关系文件要放在 assets 目录下。

（2）打开 WIFI 热点，要记得为 AndroidManifest.xml 添加网络访问的权限配置：



```
<!-- 查看网络状态 -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<!-- WLAN -->
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<!-- 上网 -->
<uses-permission android:name="android.permission.INTERNET" />
```

（3）在 AndroidManifest.xml 中注册 MAC 与设备的关系导入服务，注册代码如下：

```
<service android:name=".service.ImportService" android:enabled="true" />
```

（4）开关 WIFI 热点，只能在真机上测试，无法在模拟器上测试。

WIFI 热点的操作也是通过无线网络管理器 WifiManager 完成的。下面是 WifiManager 与 WIFI 热点有关的方法说明。

- setWifiApEnabled：开启或关闭 WIFI 热点。隐藏方法，需通过反射调用。
- getWifiApState：获取当前的 WIFI 热点状态，WIFI 热点状态的取值说明见表 14-6。

表14-6 WIFI热点状态的取值说明

WifiManager 类的 WIFI 热点状态	说明
WIFI_AP_STATE_DISABLED	WIFI 热点已关闭
WIFI_AP_STATE_DISABLING	WIFI 热点正在关闭
WIFI_AP_STATE_ENABLED	WIFI 热点已开启
WIFI_AP_STATE_ENABLING	WIFI 热点正在开启
WIFI_AP_STATE_FAILED	WIFI 热点开启失败

- isWifiApEnabled：判断 WIFI 热点是否启用。只有已连接状态才返回 true，其余都返回 false。
- getWifiApConfiguration：获取 WIFI 热点的配置信息。
- setWifiApConfiguration：设置 WIFI 热点的配置信息。
- addToBlacklist：把指定 MAC 地址添加到黑名单列表，即阻止该设备连接热点。
- clearBlacklist：清除黑名单列表。

注意上面与 WIFI 热点有关的方法都是隐藏方法，这意味着外部无法直接调用该方法。Android 因为在不断更新升级，同时新技术层出不穷，所以并没有把所有公共方法开放出来。查看 Android 的 SDK 源码会发现少数公开方法加上了 hide 标记，表示该函数是隐藏方法，尚未正式开放，原因可能是不稳定或有待完善。有时开发者确实需要调用这些隐藏方法，这时需要通过 Java 的反射机制间接实现。反射机制指的是在运行过程中，程序能够调用任意一个对象的任意公开方法和属性，而不被 hide 标记所束缚。

下面是使用反射机制实现开关 WIFI 热点与获取热点状态的代码：



```
	public static String setWifiApEnabled(WifiManager wifiMgr,WifiConfiguration apConfig,boolean
enabled) {
		String desc = "";
		if (apConfig.SSID == null || apConfig.SSID.length() <= 0) {
			desc = "WIFI 名称为空";
			return desc;
		}
		try {
			if (enabled) {
				// wifi 和 WIFI 不能同时打开，所以打开 WIFI 时需要关闭 wifi
				wifiMgr.setWifiEnabled(false);
			}
			// 通过反射调用设置 WIFI
			Method method = wifiMgr.getClass().getMethod("setWifiApEnabled",
					WifiConfiguration.class, Boolean.TYPE);
			// 返回 WIFI 打开状态
			if ((Boolean) method.invoke(wifiMgr, apConfig, enabled) != true) {
				desc = "WIFI 操作失败";
			}
		} catch (Exception e) {
			e.printStackTrace();
			desc = "WIFI 操作异常 ： " + e.getMessage();
		}
		return desc;
	}

	public static int getWifiApState(WifiManager wifiMgr) {
		try {
			Method method = wifiMgr.getClass().getMethod("getWifiApState");
			int i = (Integer) method.invoke(wifiMgr);
			if (i > 9) {
				i -= 10;
			}
			Log.d(TAG, "wifi state:  " + i);
			return i;
		} catch (Exception e) {
			e.printStackTrace();
			return WIFI_AP_STATE_FAILED;
		}
	}
```

编码完成，照例观看一下 WIFI 热点的页面效果，一开始打开热点页面，热点状态是关闭的，如图 14-35 所示。点击右上角的开关按钮，将 WIFI 热点开启，如图 14-36 所示。





图 14-35　WIFI 共享器的初始页面



图 14-36　开启 WIFI 热点功能

这个 WIFI 热点的名称是“CMAY9D***”，打开其他设备（包括手机和笔记本电脑）刷新 wifi 列表，然后点击连接新发现的 WIFI “CMAY9D***”，热点管理页面就会将该设备加入已连接的设备列表。如果是手机连接，设备名这列显示的就是手机的制造厂商；如果是笔记本连接，设备名这列显示的就是该电脑上登记的计算机名。笔者测试时，热点管理页面依次找到了一部联想手机、一部苹果手机、一部小米手机，外加一台计算机名为 OUYANGSHEN 的笔记本电脑，如图 14-37 所示。接入 WIFI 的设备一览无余，再也不用担心自己的 WIFI 被蹭了。



图 14-37　检测到已接入 WIFI 热点的设备列表

目前，WIFI 共享器主要实现了 3 个功能，即开关热点、修改热点配置和查看已连接设备信息。读者若有兴趣，可以加以完善，比如加一个小黑屋功能，把不明来源的设备加入黑名单，不让它连接本机热点；也可以加一个流量控制功能，一旦检测到热点流量超过阈值，就立即关闭 WIFI 热点，避免不必要的流量消耗。

下面是 WIFI 热点管理页面的代码：

```
public class WifiShareActivity extends AppCompatActivity implements
        OnClickListener,OnCheckedChangeListener,GetClientListener,FindNameListener {
    private static final String TAG = "WifiShareActivity";
    private CheckBox ck_wifi_switch;
    private EditText et_wifi_name, et_wifi_password;
    private Spinner sp_wifi_des;
    private TextView tv_connect;
    private LinearLayout ll_client_title;
    private ListView lv_wifi_client;
    private WifiManager mWifiManager;
    private WifiConfiguration mWifiConfig = new WifiConfiguration();
```

```
    private int mDesType = 0;
    private ArrayList<ClientScanResult> mClientArray = new ArrayList<ClientScanResult>();
    private HashMap<String, String> mapName = new HashMap<String, String>();
    private Handler mHandler = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_wifi_share);
        ck_wifi_switch = (CheckBox) findViewById(R.id.ck_wifi_switch);
        et_wifi_name = (EditText) findViewById(R.id.et_wifi_name);
        et_wifi_password = (EditText) findViewById(R.id.et_wifi_password);
        tv_connect = (TextView) findViewById(R.id.tv_connect);
        ll_client_title = (LinearLayout) findViewById(R.id.ll_client_title);
        lv_wifi_client = (ListView) findViewById(R.id.lv_wifi_client);
        findViewById(R.id.btn_wifi_save).setOnClickListener(this);
        et_wifi_name.setText(Build.SERIAL);
        et_wifi_password.setText("");
        ck_wifi_switch.setOnCheckedChangeListener(this);
        mWifiManager = (WifiManager) getSystemService(Context.WIFI_SERVICE);
        setWifiConfig();

        sp_wifi_des = (Spinner) findViewById(R.id.sp_wifi_des);
        ArrayAdapter<String> starAdapter = new ArrayAdapter<String>(this,
                R.layout.item_select, desNameArray);
        starAdapter.setDropDownViewResource(R.layout.item_select);
        sp_wifi_des.setAdapter(starAdapter);
        sp_wifi_des.setSelection(mDesType);
        sp_wifi_des.setPrompt("请选择加密方式");
        sp_wifi_des.setOnItemSelectedListener(new DesTypeSelectedListener());
        sp_wifi_des.setSelection(mDesType);
        mHandler.postDelayed(mClientTask, 50);
    }

    private String[] desNameArray = {"无", "WPA PSK", "WPA2 PSK"};
    private int[] desTypeArray = {WifiConfiguration.KeyMgmt.NONE, WifiConfiguration.KeyMgmt.
WPA_PSK, 4};
    private class DesTypeSelectedListener implements OnItemSelectedListener {
        @Override
        public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
            mDesType = arg2;
        }
```

```
        @Override
        public void onNothingSelected(AdapterView<?> arg0) {
        }
    }

    private Runnable mClientTask = new Runnable() {
        @Override
        public void run() {
            GetClientListTask getClientTask = new GetClientListTask();
            getClientTask.setGetClientListener(WifiShareActivity.this);
            getClientTask.execute();
            mHandler.postDelayed(this, 3000);
        }
    };

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_wifi_save) {
            if (et_wifi_name.getText().length() < 4) {
                Toast.makeText(this, "WIFI 名称长度需不小于四位", Toast.LENGTH_SHORT).show();
                return;
            } else if (mDesType!=0 && et_wifi_password.getText().length() < 8) {
                Toast.makeText(this, "WIFI 密码长度需不小于八位", Toast.LENGTH_SHORT).show();
                return;
            }
            Toast.makeText(this, "已保存本次 WIFI 设置", Toast.LENGTH_SHORT).show();
            //只有已开启 WIFI，才需要断开并重连。当前未开启 WIFI，保存设置后不自动开启 WIFI
            int timeout = 0;
            if (ck_wifi_switch.isChecked() == true) {
                ck_wifi_switch.setChecked(false);
                timeout = 2000;
            }
            setWifiConfig();
            mHandler.postDelayed(mReOpenTask, timeout);
        }
    }

    private void setWifiConfig() {
        mWifiConfig.allowedKeyManagement.clear();
        mWifiConfig.SSID = et_wifi_name.getText().toString();
        if (mDesType == 0) {
            mWifiConfig.preSharedKey = "";
            mWifiConfig.wepKeys[0] = et_wifi_password.getText().toString();
```

```
                mWifiConfig.wepTxKeyIndex = 0;
            } else {
                mWifiConfig.allowedKeyManagement.set(desTypeArray[mDesType]);
                mWifiConfig.preSharedKey = et_wifi_password.getText().toString();
                mWifiConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.OPEN);
                mWifiConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);
                mWifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);
                mWifiConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);
                mWifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);
                mWifiConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);
            }
        }

        private Runnable mReOpenTask = new Runnable() {
            @Override
            public void run() {
                if (WifiUtil.getWifiApState(mWifiManager) == WifiUtil.WIFI_AP_STATE_DISABLED) {
                    ck_wifi_switch.setChecked(true);
                } else {
                    mHandler.postDelayed(this, 2000);
                }
            }
        };

        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (buttonView.getId() == R.id.ck_wifi_switch) {
                String result = "";
                if (isChecked == false) {
                    result = WifiUtil.setWifiApEnabled(mWifiManager, mWifiConfig, isChecked);
                } else {
                    setWifiConfig();
                    result = WifiUtil.setWifiApEnabled(mWifiManager, mWifiConfig, isChecked);
                }
                Log.d(TAG, "onCheckedChanged: "+isChecked+". "+result);
                if (result!=null && result.length()>0) {
                    Toast.makeText(this, result, Toast.LENGTH_SHORT).show();
                    ck_wifi_switch.setChecked(!isChecked);
                }
            }
        }

        @Override
```

```
public void onGetClient(ArrayList<ClientScanResult> clientList) {
        mClientArray = clientList;
        Log.d(TAG, "mClientArray.size()=" + mClientArray.size());
        if (WifiUtil.getWifiApState(mWifiManager) != WifiUtil.WIFI_AP_STATE_ENABLING
                && WifiUtil.getWifiApState(mWifiManager) != WifiUtil.WIFI_AP_STATE_ENABLED) {
                mClientArray.clear();
        } else if (mClientArray == null) {
                mClientArray = new ArrayList<ClientScanResult>();
        }
        if (mClientArray.size() <= 0) {
                tv_connect.setText("当前没有设备连接");
                ll_client_title.setVisibility(View.GONE);
        } else {
                String desc = String.format("当前已有%d 台设备连接", mClientArray.size());
                tv_connect.setText(desc);
                ll_client_title.setVisibility(View.VISIBLE);
        }
        for (int i = 0; i < mClientArray.size(); i++) {
                ClientScanResult item = mClientArray.get(i);
                String ipAddr = item.getIpAddr();
                item.setDevice(MacManager.getInstance(this).getMacDevice(item.getHWAddr()));
                if (mapName.containsKey(ipAddr)) {
                        item.setHostName(mapName.get(ipAddr));
                } else {
                        item.setHostName(MacManager.getInstance(this).getDeviceName(item.getDevice()));
                        String upperDevice = item.getDevice().toUpperCase();
                        if (upperDevice.equals("INTEL") || upperDevice.equals("HEWLETT")
                                        || upperDevice.equals("DELL") || upperDevice.equals("ASUS")
                                        || upperDevice.equals("ACER") || upperDevice.equals("TOSHIBA")) {
                                Log.d(TAG, "new GetClientNameTask");
                                GetClientNameTask getNameTask = new GetClientNameTask();
                                getNameTask.setFindNameListener(WifiShareActivity.this);
                                getNameTask.execute(ipAddr);
                        }
                }
        }
        ClientListAdapter clientAdapter = new ClientListAdapter(this, mClientArray);
        lv_wifi_client.setAdapter(clientAdapter);
}

public static native String nameFromJNI(String ip);
public static native String unimplementedNameFromJNI(String ip);
static {
```

```
            System.loadLibrary("jni_mix");
        }

        @Override
        public void onFindName(String info) {
            if (info != null && info.length() > 0) {
                String[] split = info.split("\\|");
                if (split.length > 1 && split[1].length() > 0) {
                    mapName.put(split[0], split[1]);
                }
            }
        }
    }
```

14.5 小　　结

本章主要介绍了 App 开发用到的常见融合技术，包括网页集成（资产管理器、网页视图、简单浏览器）、JNI 开发（NDK 环境搭建、创建 JNI 接口、JNI 实现加解密）、局域网开发（无线网络管理器、蓝牙技术）。最后设计了一个实战项目“WIFI 共享器”，在该项目的 App 编码中采用了本书到目前为止的主要后台技术，实现了 WIFI 热点的共享和热点连接设备的检测。另外，介绍了 NetBIOS 协议的实际运用。

通过本章的学习，读者应该能够掌握以下 4 种开发技能：

（1）学会使用网页视图集成网页显示。
（2）学会实现 JNI 接口的编码与调用。
（3）学会使用蓝牙技术完成设备之间的数据传输。
（4）学会使用无线网络管理器进行 WIFI 热点的管理操作。





第三方开发包

手机 App 的功能日益丰富，除了 Android 系统自身不断更新换代，更离不开众多服务提供商的开发包。本章介绍 App 开发常见的第三方开发包，主要包括国内两家主要的地图服务开发（百度地图和高德地图）、全球华人主要的两个分享渠道开发（QQ 分享和微信分享）、国内两家主要的支付服务开发（支付宝和微信支付）、中文世界主要的语音服务开发（讯飞语音的语音识别和语音合成）。最后结合本章所学的知识演示一个实战项目“仿滴滴打车”的设计与实现。

15.1 地图 SDK

地图是人们日常生活中不可或缺的工具，手机上与地图有关的功能也很常见，比如定位自己在哪条街道什么位置、查查周边有哪些好吃好玩的地方等。由于地图功能与用户所在国家密切相关，因此 Android 系统自身并不提供地图功能，App 需要接入第三方地图开发包才能实现相关功能。国内常用的地图 SDK 包括百度地图和高德地图，本节对这两个地图的开发包分别进行介绍。

15.1.1 查看签名信息

尽管现在 App 的反破解手段已经很多了，不过是道高一尺、魔高一丈，各种山寨版的 App 仍然层出不穷。App 的包名相当于人们的身份证，然而这个身份证很容易被伪造，如果持有同样的身份证号，我们焉知对方是真是假？这时就要引入其他身份鉴伪标志。对于人类来说，可以通过指纹识别是否为本人。对于 App 来说，也有类似指纹的标志信息，即 App 的签名信息。如果黑客篡改了 App 的安装包，那么签名信息必然发生变化，通过校验签名就能鉴别该 App 的真伪。App 有了签名作为身份信息，才允许在 Android 系统上安装和运行。

应用一般把 SHA1 作为签名信息。在开发阶段，Android Studio 使用自带的签名文件 debug.keystore 给 App 签名；在上线阶段，开发者提供自己的签名文件给 App 做正式签名。有的第三方 SDK（如地图类的开发包）需要开发者分别提供开发版的签名和发布版的签名，以此判断 App 能否正常使用地图功能。这样一来，大家就比较关心如何才能知晓自己的 Android Studio 用的是什么签名？下面分别介绍一下开发版签名和发布版签名的获取方法。

1. 开发版签名

Android Studio 自带的签名文件位于用户目录的.android/debug.keystore。打开 Android Studio，主界面右边有一个竖排的 Gradle 按钮，单击该按钮弹出当前项目的概念结构窗口，点开项目名称内部的 Tasks/android 目录，发现其下有 3 个工具，分别是 androidDependencies、signingReport 和 sourceSets，具体的目录结构如图 15-1 所示。

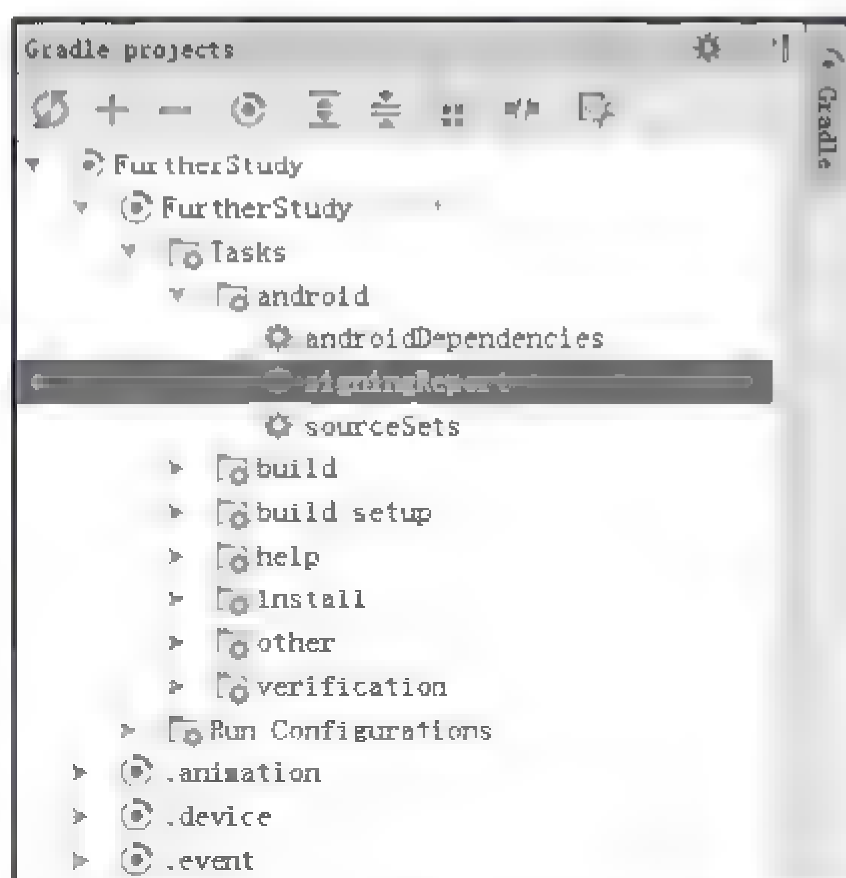


图 15-1 Gradle 项目的结构图

这里的 signingReport 为签名报告工具，双击 signingReport 运行该工具，之后 Android Studio 开始查找并报告每个模块的开发签名。报告结果打印在主界面左下方的 signingReport 窗口，框起来的 SHA1 字符串为模块 thirdsdk 的开发签名，如图 15-2 所示。



图 15-2 开发版签名的查询结果

默认的调试签名文件通常不会更改，当然也有例外情况，比如微信平台 SDK 的演示工程要求使用 demo 工程自带的签名文件。若要更换调试用的签名文件，则需要修改对应模块的 build.gradle，即在该编译文件的 android 节点下补充签名配置，表示开发版签名使用当前模块目录下的 debug.keystore。

```
signingConfigs {
    debug {
        storeFile file("debug.keystore")
    }
}
```

2. 发布版签名

第 8 章介绍 App 发布时提到使用密钥文件为 App 打包安装包，这个密钥文件就是发布版的签名文件。依次选择菜单 Build→Generate Signed APK...，在弹出的窗口中选择待打包的模块，进入 APK 签名窗口页面，如图 15-3 所示。

这里的 test.jks 为发布用的签名文件，若想查看该文件的签名信息，则可打开命令提示符窗口，在命令行输入 keytool -v -list -keystore F:\StudioProjects\test.jks，然后回车运行该命令；接着窗口提示输入密钥库口令，该口令为密钥文件的密码，输入密码并回车，稍等一会儿，命令行窗口会把该密钥文件的详细签名信息打印出来，完整的签名信息如图 15-4 所示。注意，框起来的 SHA1 字符串为发布版的签名串。

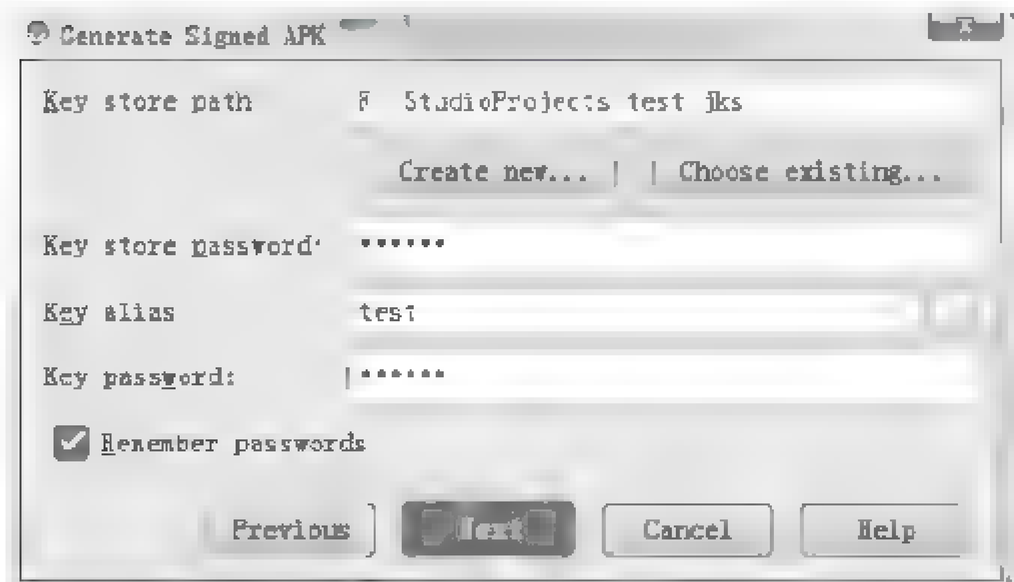


图 15-3 APK 签名窗口

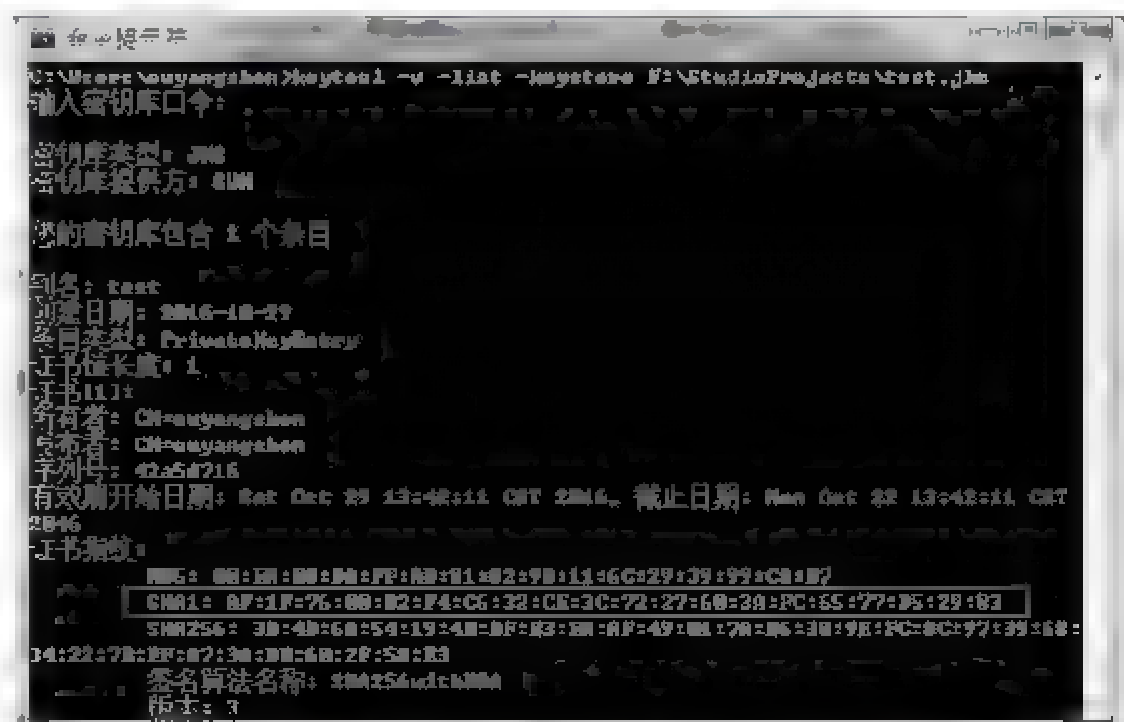


图 15-4 发布版签名的查询结果

15.1.2 百度地图

百度地图的开发网址是 http://lbsyun.baidu.com/，进入该网站后，依次选择“开发”→“Android 开发”→“Android 地图 SDK”→“相关下载”，即可打开百度地图的 SDK 下载页面。开发者可在此页面选择“自定义下载”或“一键下载”。当然，作为勤奋好学的开发者，有必要了解地图 SDK 的具体组件，这里建议选择“自定义下载”，打开的地图组件页面如图 15-5 所示。



图 15-5 百度地图 SDK 的下载页面

在该页面勾选需要集成的组件，单击页面左下方的“开发包”按钮，下载包含对应组件的地图 SDK；单击“示例代码”按钮，可下载官方的 demo 工程源代码。

有了地图 SDK，还得申请开发者账号和测试应用账号，才能在测试应用中正常使用地图功能。具体的申请步骤如下：

01 打开百度地图的开发者平台网址 http://developer.baidu.com/，依次选择“开发者中心”→“应用管理”→“创建工程”，打开应用创建页面，如图 15-6 所示。



图 15-6 百度地图的应用创建页面

02 在应用创建页面填写应用名称，然后单击“创建”按钮，创建成功后跳转到应用信息页面，如图 15-7 所示。





图 15-7 测试应用的基本信息页面

03 记下该页面“基本信息”中的 API Key，后面会用到。单击页面左侧导航栏的“LBS 服务”链接，跳转到地图服务页面，如图 15-8 所示。



图 15-8　测试应用的地图服务页面

04 在地图服务页面选择 Android SDK 应用类型，然后勾选需要启用的服务，并在下方输入测试应用的包名和 SHA1 签名串，视情况可同时填入发布版签名和开发版签名。填写完毕后单击“提交”按钮，再回到应用列表页面，一个可用的测试应用账号就申请完成了，如图 15-9 所示。

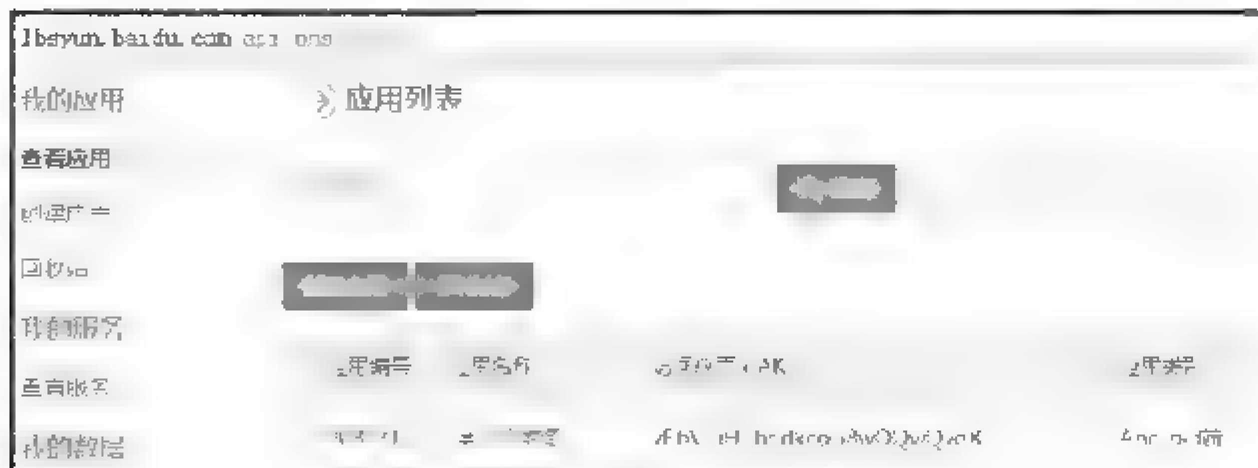


图 15-9　申请完成的应用列表页面

完成测试应用的账号申请后，接下来进行地图开发环境的搭建工作。

首先，打开 AndroidManifest.xml，在 application 节点下补充百度地图的密钥配置。其中，android:value 字段值为应用基本信息的 API Key。具体配置代码如下：

```
<!-- 百度地图密钥 -->
<meta-data
    android:name="com.baidu.lbsapi.API_KEY"
    android:value="vRbVCiHqbhdkcoG8wOQwQvdX" />
```

其次，把 SDK 包里的 BaiduLBS_Android.jar 复制到模块的 libs 目录。除 jar 文件外，把其余 so 库的所有文件夹复制到 src/main/jniLibs 目录下。

最后，把官方 demo 工程里的 com/baidu/mapapi/overlayutil 整个目录源码复制到你的工程中。该目录的源码用于 POI 搜索，原本包含在 SDK 的 jar 包中，不过百度地图 SDK3.6 及以后版本不再内置这部分代码，所以需要开发者自行将这块源码加入工程中。

好不容易搞定了地图功能的账号申请与环境搭建，终于进入大家最期待的地图开发环节了。地图的开发有很多应用场景，这里选取几个常用又相对简单的功能，方便读者快速上手。这些功能包括显示地图并定位、POI 搜索、距离与面积测量，分别介绍如下：

1. 显示地图并定位

对于地图 SDK 来说，最基础的功能是显示当前城市的地图。编码需要注意以下几点：

（1）在加载页面布局前要先对 SDK 进行初始化操作，即在 setContentView 方法之前插入下面这行代码：

```
SDKInitializer.initialize(getApplicationContext());
```

（2）一开始要先隐藏地图图层，等定位到当前城市后再开启图层显示。如果一开始默认显示北京地图，就不会直接显示当前城市的地图了。

地图相关类及对应的方法较多，且在不断更新中，无法一一列举，读者可参考百度地图官网的最新 API 说明文档。下面是有关地图显示与定位的代码：

```
private MapView mMapView;
private BaiduMap mMapLayer;
private LocationClient mLocClient;
private boolean isFirstLoc = true;                    // 是否首次定位
private double m_latitude;
private double m_longitude;

private void initLocation() {
    mMapView = (MapView) findViewById(R.id.bmapView);
    mMapView.setVisibility(View.INVISIBLE);         // 先隐藏地图，待定位到当前城市时再显示
    mMapLayer = mMapView.getMap();
    mMapLayer.setOnMapClickListener(this);
    mMapLayer.setMyLocationEnabled(true);           // 开启定位图层
```

```
        mLocClient = new LocationClient(this);
        mLocClient.registerLocationListener(new MyLocationListenner());  // 设置定位监听器
        LocationClientOption option = new LocationClientOption();
        option.setOpenGps(true);                    // 打开 GPS
        option.setCoorType("bd09ll");               // 设置坐标类型
        option.setScanSpan(1000);                   // 设置定位的时间间隔
        option.setIsNeedAddress(true);              // 设置 true 才能获得详细的地址信息
        mLocClient.setLocOption(option);            // 设置定位参数
        mLocClient.start();                         // 开始定位
    }

    public class MyLocationListenner implements BDLocationListener {
        @Override
        public void onReceiveLocation(BDLocation location) {
            if (location == null || mMapView == null) {   // map view 销毁后不再处理新接收的位置
                Log.d(TAG, "location is null or mMapView is null");
                return;
            }
            m_latitude = location.getLatitude();
            m_longitude = location.getLongitude();
            String position = String.format("当前位置 ：%s|%s|%s|%s|%s|%s|%s",
                    location.getProvince(), location.getCity(), location.getDistrict(), location.getStreet(),
                    location.getStreetNumber(), location.getAddrStr(), location.getTime());
            loc_position.setText(position);
            MyLocationData locData = new MyLocationData.Builder().accuracy(location.getRadius())
                    // 此处设置开发者获取的方向信息，顺时针 0～360
                    .direction(100).latitude(m_latitude).longitude(m_longitude).build();
            mMapLayer.setMyLocationData(locData);
            Toast.makeText(MapBaiduActivity.this, "isFirstLoc=" + isFirstLoc,
                    Toast.LENGTH_LONG).show();
            if (isFirstLoc) {
                isFirstLoc = false;
                LatLng ll = new LatLng(m_latitude, m_longitude);
                MapStatusUpdate update = MapStatusUpdateFactory.newLatLngZoom(ll, 14);
                mMapLayer.animateMapStatus(update);
                mMapView.setVisibility(View.VISIBLE);   // 定位到当前城市时再显示图层
            }
        }

        public void onReceivePoi(BDLocation poiLocation) {
        }
    }
```

百度地图定位与显示的效果如图 15-10 所示。展示的界面是笔者所在城市的地图，中央的圆点为笔者当前所处的位置。



图 15-10　百度地图定位到当前城市

2. POI 搜索

POI 即地图注点，是 Point Of Interest 的缩写，通过在地图上标注地点名称、类别、经度、纬度等信息实现携带位置信息的地图标注功能。POI 搜索是地图 SDK 的一个重要功能，根据关键词搜索并在地图上显示周边地点的查询结果，是智能出行的基础。

POI 搜索的详细代码行较多，为节约篇幅，这里就不贴出来了，读者可参考本书的下载资源。百度地图搜索 POI 的效果如图 15-11 和图 15-12 所示。其中，图 15-11 所示为输入关键词“公园”后的查询结果；点击其中某个标注，页面下方弹出小窗口提示该标注代表的公园信息，如图 15-12 所示。



图 15-11　百度地图的 POI 搜索结果



图 15-12　点击某个 POI 弹出标注信息

3. 距离与面积测量

测量距离和测量面积是地图 SDK 的一个常见功能，该功能除了在地图上添加标注外，还要用到数学中的两个公式。

其中，测距用的是勾股定理（商高定理）。勾股定理是一个基本的几何定理：一个直角三角形，两直角边的平方和等于斜边的平方。如果直角三角形两直角边为 a 和 b、斜边为 c，那么 $a^2+b^2=c^2$。

测面积用的是海伦公式（秦九韶公式）。海伦公式是利用三角形的 3 个边长直接求三角形面积，表达式为：$S=\sqrt{p(p-a)(p-b)(p-c)}$。基于海伦公式，可以推导出根据多边形各边长求多边形面积的公式，即 $S=((x_0y_1-x_1y_0)+(x_1y_2-x_2y_1)+\ldots+(x_ny_0-x_0y_n))/2$。

进行测量时，还要在地图上添加标记，如一条线段的两个顶点、一个多边形的各个顶点，由此衍生各种形状的添加方式。调用 MapLayer 对象的 addOverlay 方法即可在地图上添加标记，可添加的标记形状说明见表 15-1。

表15-1 百度地图的标记说明

百度地图的标记类	MapLayer 类的添加方法	说明
ArcOptions	addOverlay	弧线
CircleOptions	addOverlay	圆圈
MarkerOptions	addOverlay	图片
PolygonOptions	addOverlay	多边形
PolylineOptions	addOverlay	线段
TextOptions	addOverlay	文本

弄懂了测量的算法原理和在地图上添加标记的方法，测距与测面积的实现就不难了。为节省篇幅，这里不再贴出距离与面积测量的代码，读者可自行查看本书下载资源中的代码。

使用百度地图测距与测面积的效果如图 15-13 和图 15-14 所示。其中，图 15-13 展示了森林公园与西湖的测距结果，可以看到两点之间距离 5.9 千米。再来看面积测量的结果，图 15-14 显示西湖公园的面积大约是 84 万平方米。



图 15-13 百度地图的距离测量结果

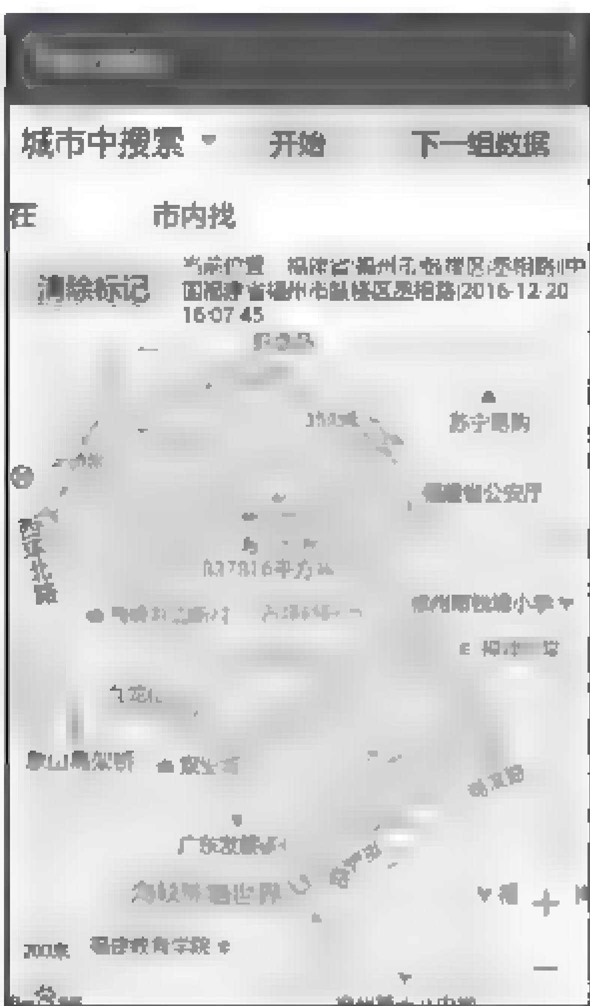


图 15-14 百度地图的面积测量结果

15.1.3 高德地图

高德地图的开发网址是 http://lbs.amap.com/，进入该网站后，依次选择“开发”→“Android 平台”→“Android 地图 SDK”，将页面拉到底，单击左下方的“相关下载”链接，打开下载页面，如图 15-15 所示。

单击下载页面的“自定义下载”按钮，向下拉出组件列表，勾选需要下载的组件与资料，然后单击“下载”按钮开始下载地图 SDK 与示例代码。

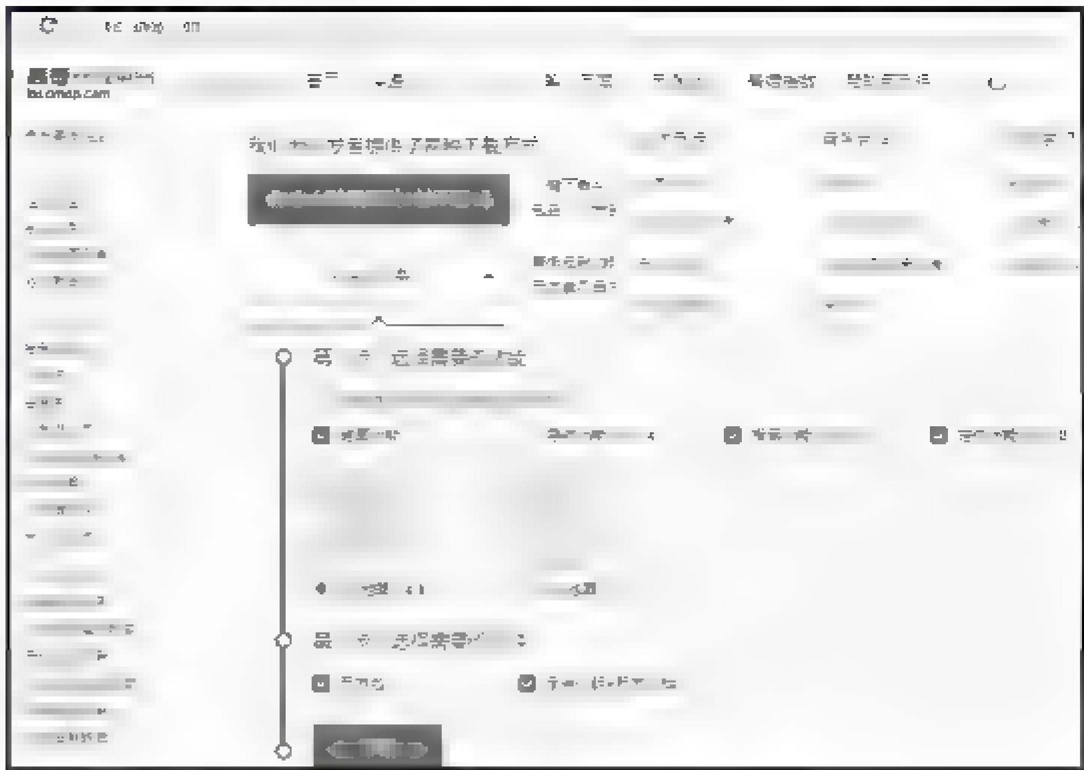


图 15-15　高德地图 SDK 的下载页面

高德地图也需要申请开发者账号和测试应用账号，使用开发者账号登录后，即可在网页右上角找到“控制台”链接，依次单击“控制台”→“创建新应用”，弹出“创建应用”窗口，如图 15-16 所示。



图 15-16　高德地图的“创建应用”窗口

填写应用名称并选择应用类型，然后单击“创建”按钮，即可看到应用列表增加了一条刚创建的应用记录，如图 15-17 所示。



图 15-17　测试应用的初始信息

单击应用记录右边的“添加新 Key”按钮，弹出“为第三方应用添加 Key”窗口，在此可设置应用的 Key 名称、SHA1 签名、包名等信息，如图 15-18 所示。

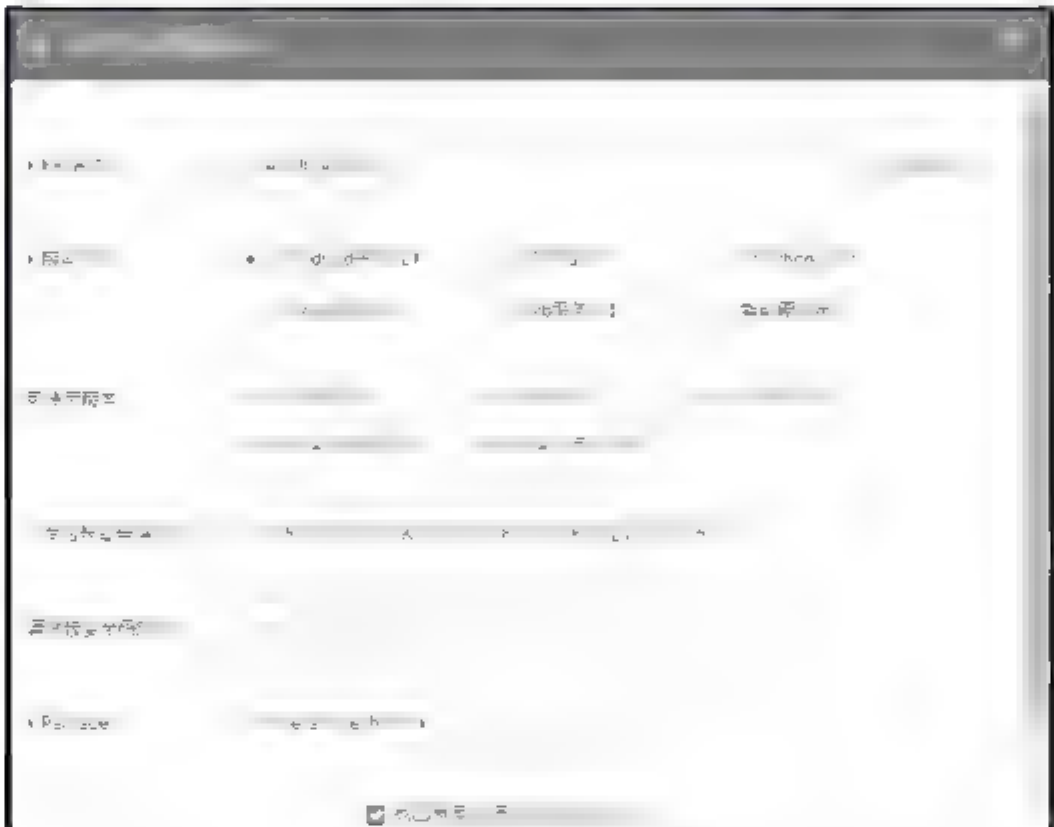


图 15-18　“为第三方应用添加 Key”窗口

在设置窗口填写测试应用的 Key 名称，选中服务平台 Android 平台 SDK，并分别填写发布版签名、调试版签名（开发签名）、Package（包名），注意勾选“我已阅读***”，然后单击“提交”按钮完成设置操作。回到应用列表页面，此时测试应用下面多了一条刚添加的 Key 记录，如图 15-19 所示。记下这里的 Key 值，后面会用到。

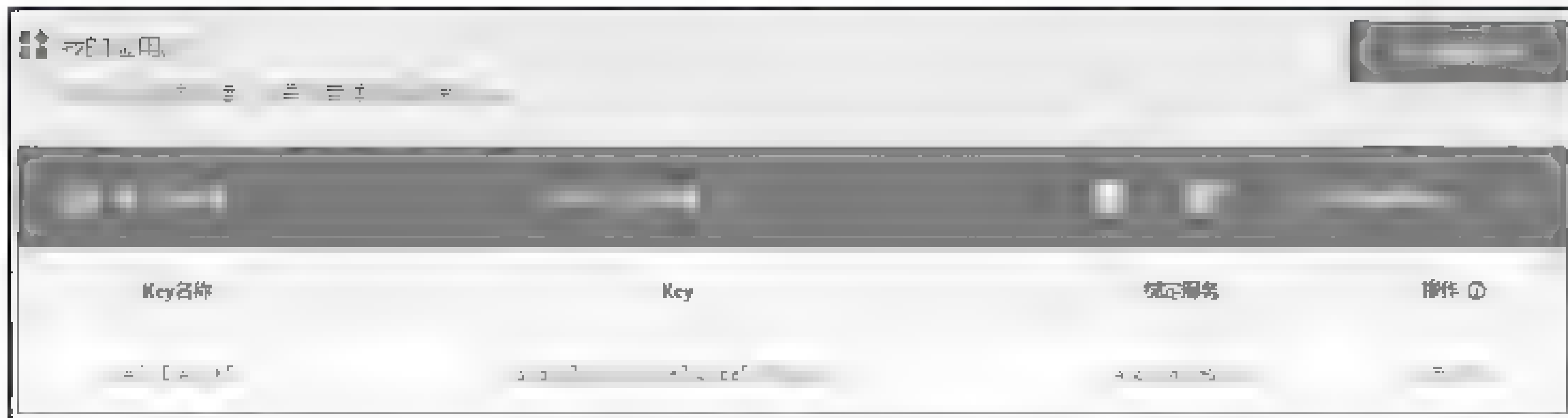


图 15-19　测试应用的键值信息

测试应用账号申请完成，继续搭建高德地图的开发环境。

首先，打开 AndroidManifest.xml，在 application 节点下补充高德地图的密钥配置。其中，android:value 字段值为测试应用的 Key 值。具体配置代码如下：

```
<!-- 高德地图密钥 -->
<meta-data
    android:name="com.amap.api.v2.apikey"
    android:value="d2d982826l5cb90e78b4be537636647c" />
```

同时还要注册高德地图的定位服务，具体的服务注册代码如下：

```
<service android:name="com.amap.api.location.APSService" />
```

其次，把 SDK 包里的 AMap***.jar 复制到模块的 libs 目录，JAR 文件可能只有一个，也可能有多个，如 AMap2DMap***.jar、AMapSearch***.jar、AMapLocation***.jar 等，如此便完成了高德地图的开发环境搭建工作。

下面介绍高德地图的 3 个主要功能：显示地图并定位、POI 搜索、距离与面积测量。

1. 显示地图并定位

高德地图也要先隐藏地图图层，等到定位到当前城市后再开启图层显示。具体的定位代码如下：

```
private MapView mMapView;
private AMap mMapLayer;
private AMapLocationClient mLocClient;
private boolean isFirstLoc = true;// 是否首次定位
private double m_latitude;
private double m_longitude;

private void initLocation(Bundle savedInstanceState) {
    mMapView = (MapView) findViewById(R.id.amapView);
    mMapView.onCreate(savedInstanceState);
```

```
        mMapView.setVisibility(View.INVISIBLE);  // 先隐藏地图，待定位到当前城市时再显示
        if (mMapLayer == null) {
            mMapLayer = mMapView.getMap();
        }
        mMapLayer.setOnMapClickListener(this);
        mMapLayer.setMyLocationEnabled(true);  // 开启定位图层
        mLocClient = new AMapLocationClient(this.getApplicationContext());
        mLocClient.setLocationListener(new MyLocationListenner());  // 设置定位监听器
        AMapLocationClientOption option = new AMapLocationClientOption();
        option.setLocationMode(AMapLocationMode.Battery_Saving);
        option.setNeedAddress(true);  // 设置 true 才能获得详细的地址信息
        mLocClient.setLocationOption(option);  // 设置定位参数
        mLocClient.startLocation();  // 开始定位
    }

    public class MyLocationListenner implements AMapLocationListener {
        @Override
        public void onLocationChanged(AMapLocation location) {
            if (location == null || mMapView == null) {  // map view 销毁后不再处理新接收的位置
                Log.d(TAG, "location is null or mMapView is null");
                return;
            }
            m_latitude = location.getLatitude();
            m_longitude = location.getLongitude();
            String position = String.format("当前位置 ：%s|%s|%s|%s|%s|%s|%s",
                    location.getProvince(), location.getCity(), location.getDistrict(),
location.getStreet(),
                    location.getAdCode(), location.getAddress(), location.getTime());
            loc_position.setText(position);
            if (isFirstLoc) {
                isFirstLoc = false;
                LatLng ll = new LatLng(m_latitude, m_longitude);
                CameraUpdate update = CameraUpdateFactory.newLatLngZoom(ll, 12);
                mMapLayer.moveCamera(update);
                mMapView.setVisibility(View.VISIBLE);  // 定位到当前城市时再显示图层
            }
        }
    }
```

高德地图定位与显示的效果如图 15-20 所示，展示的界面是笔者所在城市的地图，笔者当前所处的位置在地图正中央。

2. POI 搜索

高德地图搜索 POI 的流程与百度地图类似，具体代码参见本书的下载资源。POI 搜索的效果如图 15-21 和图 15-22 所示。其中，图 15-21 所示为输入关键词“公园”后的查询结果；点击其中某个标注，标注上方弹出小窗口，提示该标注代表的公园信息，如图 15-22 所示。



图 15-20 高德地图定位到当前城市　图 15-21 高德地图的 POI 搜索结果　图 15-22 点击某个 POI 弹出标注信息

3. 距离与面积测量

在高德地图上添加标记也是通过调用 MapLayer 对象的 add*** 方法，可添加的标记和对应的方法说明见表 15-2。

表15-2 高德地图的标记说明

高德地图的标记类	MapLayer 类的添加方法	说明
CircleOptions	addCircle	圆圈
MarkerOptions	addMarker	图片
PolygonOptions	addPolygon	多边形
PolylineOptions	addPolyline	线段
TextOptions	addText	文本

使用高德地图测距与测面积的效果如图 15-23 和图 15-24 所示。其中，图 15-23 展示了福州火车站与国家 5A 景区三坊七巷的测距结果，可以看到两点之间距离 4.0 千米。另外，再看看测量岛屿面积的结果，图 15-24 显示闽江口琅岐岛的面积大约是 59 平方公里，与官方公布的岛屿陆地面积 55 平方公里相差不远。

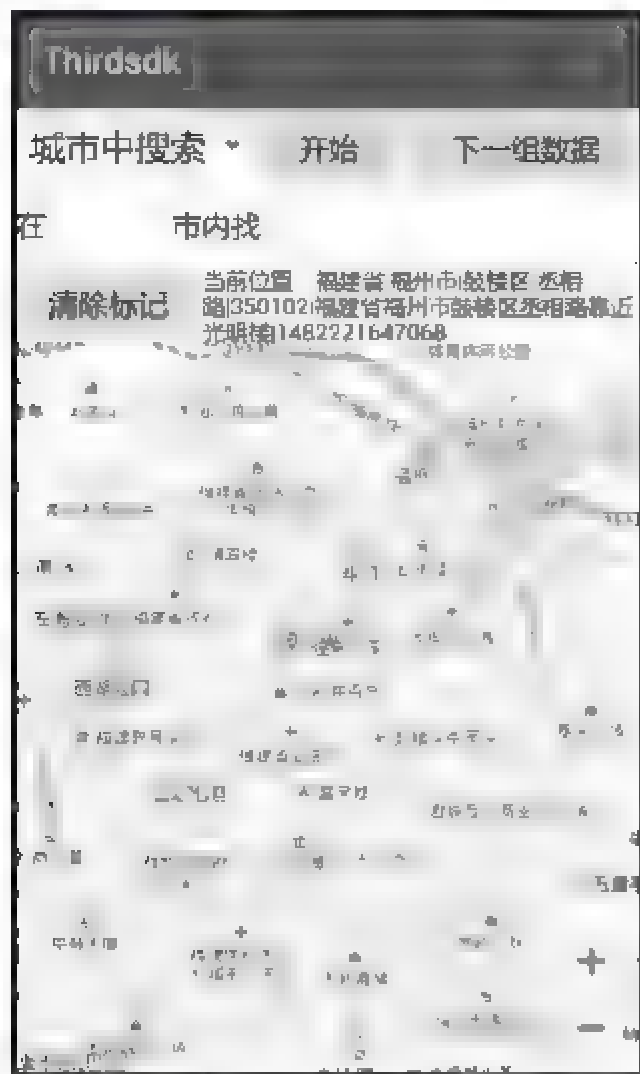


图 15-23　高德地图的距离测量结果

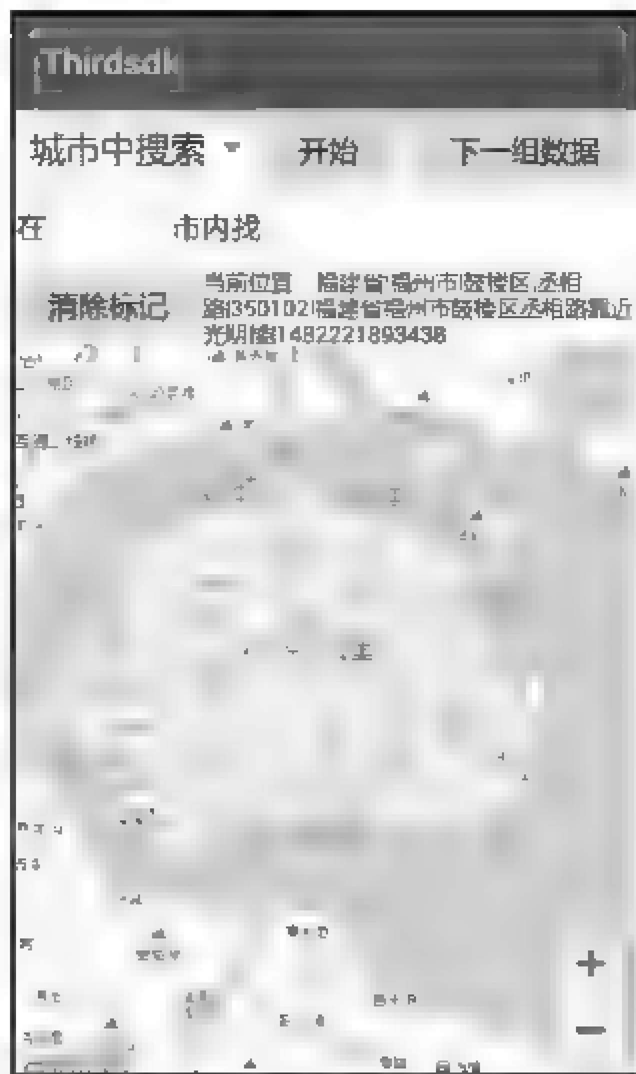


图 15-24　高德地图的面积测量结果

15.2　分享 SDK

社会化分享指的是用户通过互联网这个媒介把文本/图片/多媒体信息分享到该用户的交际圈，从而加快信息传播的行为。对于 App 来说，网络社区虽多，但用户量足够大的就那么几个，App 的社会化分享功能抓住几个大的圈子就够了，比如 QQ、微信、QQ 空间、微信朋友圈等。本节介绍 QQ 分享与微信分享的实现方案。

15.2.1　QQ 分享

QQ 好友分享与 QQ 空间分享同属 QQ 互联平台上的 QQ 分享，该平台的网址是 https://connect.qq.com/。依次单击平台首页的“文档资料”→左边导航栏的“SDK 及资源下载”→“SDK 下载页面”，进入 QQ 分享的 SDK 下载页面。下载页面上的说明资料比较详细，这里主要介绍与 QQ 分享相关的方法与参数。

下面是 QQ 分享用到的 Tencent 类的主要方法说明。

- createInstance：根据 appid 创建一个 Tencent 实例。
- login：QQ 账号登录。该方法需指定登录回调监听器 IUiListener。
- setAccessToken：设置入口令牌。登录成功后设置，即完成授权动作。
- setOpenId：设置开放标识。登录成功后设置，即完成授权动作。
- getQQToken：获取 QQ 登录授权的令牌。分享到腾讯微博时才需使用该方法。
- shareToQQ：分享给 QQ 好友。该方法需指定分享参数，分享参数的取值说明见表 15-3。
- shareToQzone：分享到 QQ 空间。该方法需指定分享参数，分享参数的取值说明见表 15-3。

表15-3　QQ分享的接口参数说明

QQShare 类的分享参数	说明
SHARE_TO_QQ_KEY_TYPE	分享类型。图文分享（普通分享）填 Tencent.SHARE_TO_QQ_TYPE_DEFAULT
PARAM_TARGET_URL	分享消息被点击后的跳转 URL
PARAM_TITLE	分享的标题
PARAM_SUMMARY	分享的消息摘要
SHARE_TO_QQ_IMAGE_URL	分享图片的 URL 或本地路径
SHARE_TO_QQ_APP_NAME	在手机 QQ 顶部的“返回”按钮文字后加上应用名。若为空，则“返回”按钮保持原样

QQ 分享完毕后可能收不到回调事件，这是因为有的手机会自动回收资源。要想避免该问题，得重写 Activity 页面的 onActivityResult 方法，加入 Tencent 类的 onActivityResultData 方法调用，示例代码如下：

```
        @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            Log.d(TAG, "-->onActivityResult " + requestCode   + " resultCode=" + resultCode);
            if (requestCode==Constants.REQUEST_LOGIN || requestCode==Constants.REQUEST_APPBAR) {

    Tencent.onActivityResultData(requestCode,resultCode,data,ItemTencentAdapter.mLoginListener);
            } else if (requestCode==Constants.REQUEST_QQ_SHARE || requestCode==Constants.
REQUEST_QZONE_SHARE) {
             Tencent.onActivityResultData(requestCode,resultCode,data,ItemTencentAdapter.mShareListener);
            }
            super.onActivityResult(requestCode, resultCode, data);
        }
```

QQ 分享的效果如图 15-25～图 15-28 所示。其中，图 15-25 所示为待分享信息的标题与内容文本；点击分享按钮弹出分享渠道窗口，如图 15-26 所示，当前支持 QQ 好友、QQ 空间、腾讯微博 3 个渠道的分享；单击 QQ 好友图标跳转到发送页面，如图 15-27 所示，可在此选择消息分享的好友对象；选择分享的对象好友后可在聊天消息窗口看到分享内容，如图 15-28 所示，包含分享的标题、内容与图片等信息。

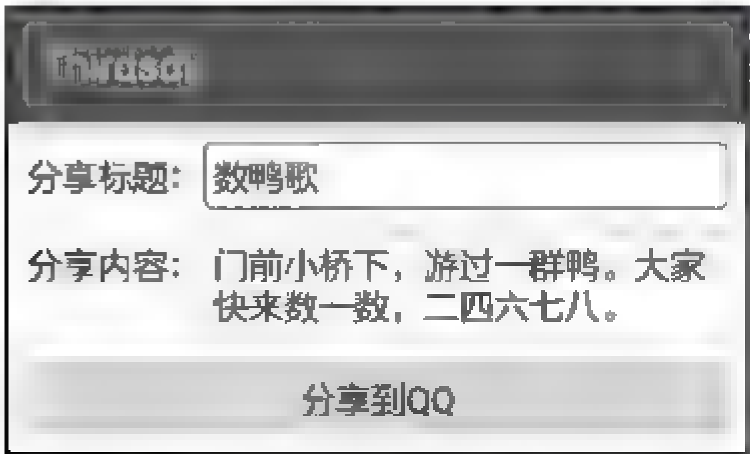


图 15-25　待分享的消息内容



图 15-26　QQ 分享的渠道列表



图 15-27　选择分享的好友对象

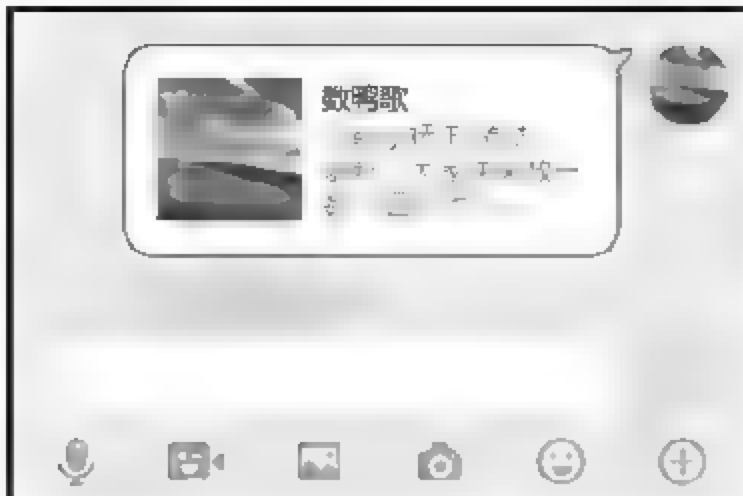


图 15-28 分享完成的聊天消息

15.2.2　微信分享

尽管微信与 QQ 都是腾讯公司开发，不过它们各自有自己的开放平台。微信开放平台的网址是 https://open.weixin.qq.com/，在平台首页依次单击链接“资源中心”→左边导航栏“资源下载”→“Android 资源下载”，即可在打开的页面中下载开发工具包与范例代码。使用范例代码演示时，注意修改以下 3 处地方：

1. 将模块的开发签名文件设置为 demo 工程自带的 debug.keystore

打开模块的编译文件 build.gradle，在该文件的 android 节点下补充签名配置，具体的配置代码如下：

```
signingConfigs {
    debug {
        storeFile file("debug.keystore")
    }
}
```

2. 将包名改为 demo 工程的包名 net.sourceforge.simcpux

除了 AndroidManifest.xml 的 package 节点值需要更改外，还要修改 build.gradle 里面的包名配置，即将 applicationId 值改为新的包名，具体的配置代码如下：

```
defaultConfig {
    applicationId "net.sourceforge.simcpux"
    minSdkVersion 15
    targetSdkVersion 25
    versionCode 1
    versionName "1.0"
}
```

3. 在 AndroidManifest.xml 中注册微信分享的回调页面 WXEntryActivity

WXEntryActivity.java 文件必须位于“包名.wxapi”这个包下面，否则无法正确收到微信分享的返回结果。同时要在 AndroidManifest.xml 中注册该活动页面，具体的注册代码如下：

```
<activity
    android:name="net.sourceforge.simcpux.wxapi.WXEntryActivity"
    android:configChanges="keyboardHidden|orientation|screenSize"
    android:exported="true"
    android:screenOrientation="portrait"
    android:theme="@android:style/Theme.Translucent.NoTitleBar" />
```

微信好友分享与微信朋友圈分享统称为微信分享，主要用到 IWXAPI、SendMessageToWX.Req 和 WXMediaMessage 三个类。下面是 IWXAPI 的常用方法说明。

- createWXAPI：创建一个微信 API 实例。当传入的 appid 为空时，还需调用 registerApp 方法进行注册；注册完毕后再传入 appid，此时获得的实例才可进行后续分享。
- registerApp：注册指定的 appid。
- sendReq：发送分享请求。该方法的参数为 SendMessageToWX.Req 对象。

下面是 SendMessageToWX.Req 的常用属性说明。

- transaction：本次请求的流水。用于标识每次请求的唯一性。
- scene：本次请求的场景。SendMessageToWX.Req.WXSceneSession 表示分享给微信好友，SendMessageToWX.Req.WXSceneTimeline 表示分享到朋友圈。
- message：本次请求的信息。该方法的参数为 WXMediaMessage 对象。

下面是 WXMediaMessage 的常用属性说明。

- title：分享的标题。
- description：分享的内容。
- mediaObject：分享的媒体信息。媒体信息的对象说明见表 15-4。
- thumbData：分享的缩略图。

表15-4 微信分享的媒体对象说明

媒体对象类	说明
WXTextObject	文本
WXImageObject	图片
WXWebpageObject	图文（既有文本，又有图片）
WXMusicObject	音乐
WXVideoObject	视频
WXFileObject	文件

QQ 分享与微信分享的使用代码片段如下：

```
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3) {
    ShareChanels item = mChannelList.get(arg2);
    mHandler.sendEmptyMessageDelayed(0, 1500);
    if (item.channelType == WEIXIN) {
        SendMessageToWX.Req req = new SendMessageToWX.Req();
```

```
                // transaction 字段用于唯一标识一个请求
                req.transaction = "wx_share" + System.currentTimeMillis();
                req.message = getWXMessage();
                req.scene = SendMessageToWX.Req.WXSceneSession;
                mWeixinApi.sendReq(req);
            } else if (item.channelType == CIRCLE) {
                SendMessageToWX.Req req = new SendMessageToWX.Req();
                req.transaction = "wx_share" + System.currentTimeMillis();
                req.message = getWXMessage();
                req.scene = SendMessageToWX.Req.WXSceneTimeline;
                mWeixinApi.sendReq(req);
            } else if (item.channelType == QQ) {
                mShareListener = new ShareQQListener(mContext, item.channelName);
                Bundle params = new Bundle();
                params.putInt(QQShare.SHARE_TO_QQ_KEY_TYPE, QQShare.SHARE_TO_QQ_
TYPE_DEFAULT);
                params.putString(QQShare.SHARE_TO_QQ_TITLE, mTitle);
                params.putString(QQShare.SHARE_TO_QQ_SUMMARY, mContent);
                params.putString(QQShare.SHARE_TO_QQ_TARGET_URL, mUrl);
                params.putString(QQShare.SHARE_TO_QQ_IMAGE_URL, mImageUrl);
                params.putString(QQShare.SHARE_TO_QQ_APP_NAME, mContext.getPackageName());
                mTencent.shareToQQ((Activity) mContext, params, mShareListener);
            } else if (item.channelType == QZONE) {
                mShareListener = new ShareQQListener(mContext, item.channelName);
                ArrayList<String> urlList = new ArrayList<String>();
                urlList.add(mImageUrl);
                Log.d(TAG, "mImageUrl=" + mImageUrl);
                Bundle params = new Bundle();
                params.putInt(QzoneShare.SHARE_TO_QZONE_KEY_TYPE, QzoneShare.SHARE_TO_
QZONE_TYPE_IMAGE_TEXT);
                params.putString(QzoneShare.SHARE_TO_QQ_TITLE, mTitle);
                params.putString(QzoneShare.SHARE_TO_QQ_SUMMARY, mContent);
                params.putString(QzoneShare.SHARE_TO_QQ_TARGET_URL, mUrl);
                params.putStringArrayList(QzoneShare.SHARE_TO_QQ_IMAGE_URL, urlList);
                Log.d(TAG, "begin shareToQzone");
                mTencent.shareToQzone((Activity) mContext, params, mShareListener);
                Log.d(TAG, "end shareToQzone");
            } else if (item.channelType == WEIBO) {   // 腾讯微博分享需要 QQ 登录授权
                mTencent.login((Activity) mContext, "all", mLoginListener);
            }
        }

    private int THUMB_SIZE = 150;
```



```
    private WXMediaMessage getWXMessage() {
        WXMediaMessage msg = new WXMediaMessage();
        if (!TextUtils.isEmpty(mTitle) && TextUtils.isEmpty(mImageUrl)) {   // 分享文本消息
            WXTextObject textObj = new WXTextObject();
            textObj.text = mContent;
            msg.mediaObject = textObj;
            msg.title = mTitle;
            msg.description = mContent;
        } else if (TextUtils.isEmpty(mTitle) && !TextUtils.isEmpty(mImageUrl)) {   // 分享图片消息
            Bitmap bmp = BitmapFactory.decodeFile(mImageUrl);
            WXImageObject imgObj = new WXImageObject(bmp);
            msg.mediaObject = imgObj;
            Bitmap thumbBmp = Bitmap.createScaledBitmap(bmp, THUMB_SIZE, THUMB_SIZE,
true);
            msg.thumbData = CacheUtil.bmpToByteArray(thumbBmp, true); // 设置缩略图
        } else if (!TextUtils.isEmpty(mTitle) && !TextUtils.isEmpty(mImageUrl)) {   // 分享图文消息
            WXWebpageObject webpage = new WXWebpageObject();
            webpage.webpageUrl = mUrl;
            msg.title = mTitle;
            msg.description = mContent;
            msg.mediaObject = webpage;
            Bitmap bmp = BitmapFactory.decodeFile(mImageUrl);
            Bitmap thumbBmp = Bitmap.createScaledBitmap(bmp, THUMB_SIZE, THUMB_SIZE,
true);
            msg.thumbData = CacheUtil.bmpToByteArray(thumbBmp, true); // 设置缩略图
        }
        return msg;
    }

    public static ShareQQListener mShareListener;
    private static class ShareQQListener implements IUiListener {
        private Context context;
        private String cn;
        public ShareQQListener(final Context context, final String channelName) {
            this.context = context;
            this.cn = channelName;
        }

        @Override
        public void onComplete(Object object) {
            Toast.makeText(context, cn + "分享完成:" + object.toString(), Toast.LENGTH_LONG).show();
        }
```

```
            @Override
            public void onError(UiError error) {
                Toast.makeText(context, cn + "分享失败:" + error.errorMessage, Toast.LENGTH_LONG).show();
            }

            @Override
            public void onCancel() {
                Toast.makeText(context, cn + "分享取消", Toast.LENGTH_LONG).show();
            }
        }
```

微信分享的效果如图 15-29～图 15-32 所示。其中，图 15-29 所示为待分享信息的标题与内容文本；点击分享按钮弹出分享渠道窗口，如图 15-30 所示，当前支持包括微信好友、微信朋友圈在内的 5 个渠道分享；点击微信好友图标跳转到好友选择页面，如图 15-31 所示，可在此选择消息分享的好友对象；选择分享的对象好友后可在聊天消息窗口看到分享内容，如图 15-32 所示，包含分享的标题、内容与图片等信息。

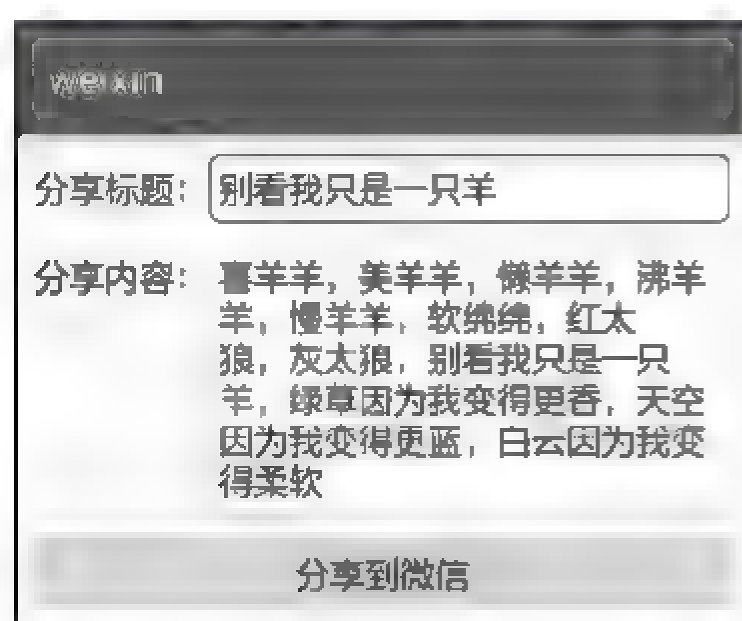


图 15-29　待分享的消息内容



图 15-30　微信分享的渠道列表



图 15-31　选择分享的好友对象



图 15-32　分享完成的聊天消息

15.3　支付 SDK

第三方支付指的是第三方平台与各银行签约，在买方与卖方之间实现中介担保，从而增强支付交易的安全性。国内常用的支付平台主要有支付宝和微信支付。其中，支付宝的市场份额为 71.5%，微信支付的市场份额为 15.99%。也就是说，这两家垄断了 7/8 的支付市场（2015 年数据）。本节对支付宝和微信支付分别进行介绍。

15.3.1 支付宝支付

因为第三方支付只是一个中介，交易流程要多次确认，所以 App 若要集成支付 SDK，需要进行以下处理：

（1）除了作为买方的用户自己拥有支付账号，开发者还得申请作为卖方的商户账号。

（2）支付过程中，虽然允许 App 直接与第三方支付平台通信，但是正常要有自己的后台服务器，由服务器与第三方平台进行通信。这样做的好处是，一方面自己后台掌握了用户交易记录，做账有依据，管理也方便；另一方面，关键交易在服务器处理，减少了恶意篡改的风险。

（3）为保证信息安全，需对关键数据进行加密处理，如支付宝采用 RSA+BASE64 算法，微信支付采用 MD5 算法。

支付宝的官方平台是蚂蚁金服开放平台，网址是 https://open.alipay.com/。在平台首页依次单击"文档中心"→左边导航栏的"资源下载"→"开发工具包下载"→"App 支付 DEMO&SDK"，在打开的页面中点击下载支付宝 SDK 及其 DEMO 工程。

另外，申请商户账号需要创建测试应用，在蚂蚁金服平台登录成功后，依次单击"研发管理"→"创建应用"，填写应用相关信息，提交成功后返回应用列表页面。然后查看应用的详情页，单击"应用环境"链接，在环境页面设置 RSA 密钥，如图 15-33 所示。记下该应用的 APPID，后面会用到。



图 15-33 支付宝测试应用的环境设置页面

集成支付宝 SDK 比较简单，除了必要的权限外，无须修改 AndroidManifest.xml，JAR 包也只要导入 alipaySdk-***.jar 即可。前面商户账号的申请信息有几个会在代码中体现，包括商户收款账号（开发者的支付宝账号）、商户的合作编号（测试应用的 APPID）、商户的 RSA 私钥（在应用环境页面中设置的 RSA 密钥）。

使用支付宝 SDK 的交易流程大致如下：

（1）按照指定格式封装好交易信息。
（2）对交易信息进行 RSA 加密与 URL 编码。
（3）调用支付接口，传入加密好的信息串（这步要另开线程处理，不能放在 UI 线程中）。
（4）支付宝 SDK 在界面下方弹出支付窗口，用户输入支付账号信息，提交支付。
（5）收到支付完成的结果，判断支付状态是成功还是失败，并做相应的后续处理。

具体的编码实现方面，支付宝官方的 DEMO 工程采用了 Thread+Handler 的异步处理模式，不过该模式要把线程代码写在 Activity 页面中，不便管理与后续维护，因此笔者的演示代码将其改造为 AsyncTask 异步任务处理方式，详细代码内容参见本书的下载资源。

支付宝 SDK 的演示效果如图 15-34 和图 15-35 所示。其中，图 15-34 所示为待付费的商品详情；点击“支付宝支付”按钮，页面下方弹出对话框，等待用户确认付款，如图 15-35 所示。



图 15-34　待支付的商品信息



图 15-35　支付宝的付款弹窗

15.3.2　微信支付

微信支付的官方平台是微信开放平台，网址是 https://open.weixin.qq.com/。在平台首页依次单击“资源中心”→左边导航栏的“资源下载”→“Android 资源下载”，即可在打开的页面中下载开发工具包与范例代码，注意这里的开发包 libammsdk.jar 同时集成了微信分享与微信支付的 SDK。

使用微信支付也需先申请测试应用，在微信开放平台登录成功后，依次单击链接“管理中心”→“创建移动应用”，填写应用相关信息，提交成功后返回应用列表页面。然后查看应用的详情页，在接口信息栏目中发现默认已获得微信分享的权限，而微信支付权限需要另外申请开通，如图 15-36 所示。

因为个人开发者无法申请微信支付功能，所示只能使用官方 DEMO 工程里的测试账号进行演示。由于微信支付与微信分享在同一个开发包中，因此集成步骤与微信分享大致相同，额外需要注意以下两点：



图 15-36　微信平台测试应用的接口信息页面

（1）支付结果页面的代码 WXPayEntryActivity.java 必须放在“包名.wxapi”这个包下面。另外，AndroidManifest.xml 也要补充注册，activity 节点的注册配置举例如下：

```
<!-- 微信支付回调页面 -->
<activity
    android:name="net.sourceforge.simcpux.wxapi.WXPayEntryActivity"
    android:exported="true"
    android:launchMode="singleTop" >
</activity>
```

（2）确保测试设备安装了微信，并且已有默认登录的微信账号。如果设备上没有安装微信，那么在调用微信支付时会报错 Failed to find provider info for com.tencent.mm.sdk.plugin.provider。

使用微信支付的交易流程大致如下：

（1）使用开发者申请到的 APP_ID 和 APP_SECRET 向微信平台请求获取入口令牌。
（2）封装订单信息（使用开发者申请到的 PARTNER_ID 和 PARTNER_KEY），并对订单信息进行 MD5 摘要处理。
（3）把加密后的订单与入口令牌发给微信平台，生成预支付订单，返回预付订单编号。
（4）重新封装订单信息，加上预付订单编号，向微信平台发起支付交易。
（5）微信 SDK 跳到微信支付页面，用户输入支付账号信息，提交支付。
（6）支付完成，回到支付结果页面，根据处理结果进行回调操作。

编码方面，微信支付与支付宝一样建议把支付操作交给商户的后台服务器运行，不要由 App 直接与支付平台进行付款交易，演示工程里的测试代码只是为了说明交互的流程，不可作为正式支付应用。

微信支付 SDK 的演示效果如图 15-37 和图 15-38 所示。其中，图 15-37 所示为待付费的商品详情；点击“微信支付”按钮后，跳转到微信支付的交易页面，等待用户确认付款，如图 15-38 所示。



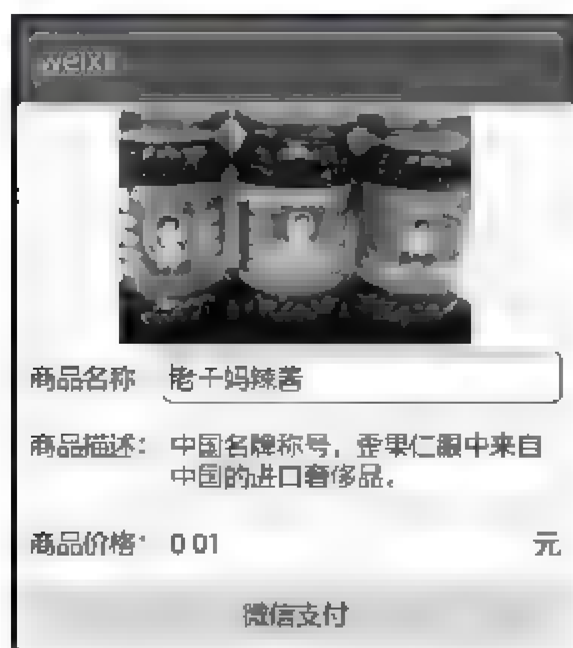


图 15-37 待支付的商品信息



图 15-38 微信支付的付款页面

15.4 语音 SDK

如今，越来越多 App 用到了语音播报功能，如地图导航、天气预报、文字阅读、口语训练等。语音技术主要分为两块，一块是语音转文字，即语音识别；另一块是文字转语音，即语音合成。国内的语音服务提供商主要有两家：讯飞语音和百度语音。本节主要介绍讯飞语音的语音识别和语音合成功能。

15.4.1 语音识别

讯飞语音的开放平台网址是 http://www.xfyun.cn/。在平台首页单击“SDK 下载”链接，在下载页面选择服务（语音听写和在线语音合成）、平台（选择 Android）、选择应用（一开始要创建新应用），然后单击“下载 SDK”按钮，等待下载开发包。

注意讯飞语音在下载 SDK 前要先创建应用，不妨把应用创建操作提到前面来。开发者在讯飞开放平台注册并成功登录后，依次单击链接“控制台”→左边导航栏的“创建新应用”，打开应用创建页面，如图 15-39 所示。



图 15-39 讯飞语音的应用创建页面

填写各项应用信息，并勾选“我已阅读并接受***”，然后单击“提交”按钮，回到应用信息页面，如图 15-40 所示。



图 15-40　测试应用的初始信息页面

应用刚创建完默认未开通任何服务，因此需要单击应用页面上的“立即开通”链接，弹出选择开通业务窗口，如图 15-41 所示。

首先勾选“语音听写”，单击“确定”按钮开通语音听写服务。因为每次只能开通一项服务，所以回到应用信息页面后，单击“开通更多服务”链接，再次打开业务开通窗口，然后勾选“在线语音合成”，并单击“确定”按钮开通语音合成服务，如图 15-42 所示。



图 15-41　开通语音识别服务



图 15-42　开通语音合成服务

语音听写和语音合成服务都申请开通后，回到应用信息页面，即可看到该测试应用的已开通服务列表已包含这两项，如图 15-43 所示。同时记下测试应用的 Appid，后面会用到。



图 15-43　开通服务后的应用信息页面

集成讯飞语音 SDK 需要注意以下几点：

（1）将 Msc.jar、Sunflower.jar 导入 libs 目录，将 libmsc.so 整个目录导入 src/main/jniLibs。（注意这些文件必须来自对应的 SDK，如果用别的 SDK，运行就会报错“用户校验失败”）。

（2）自定义一个 Application 类，在 onCreate 函数中加入以下代码，注意 appid 值为创建测试应用时分配到的 Appid：

```
SpeechUtility.createUtility(MainApplication.this, "appid=58561727");
```

（3）在 AndroidManifest.xml 中加入必要的权限和自定义的 Application 类。

（4）如果使用了 RecognizerDialog 控件，就要把 DEMO 工程 assets 目录下的文件原样复制过来。

（5）在混淆打包时需要添加-keep class com.iflytek.**{*;}，避免混淆导致 SDK 不可用。

讯飞 SDK 的语音识别功能主要通过 SpeechRecognizer 类实现，看以下常用方法。

- createRecognizer：创建语音识别对象。
- setParameter：设置语音识别的参数。语音识别的参数说明见表 15-5。

表15-5　语音识别的参数说明

SpeechConstant 类的识别参数	说明
ENGINE_TYPE	设置听写引擎。TYPE_LOCAL 表示本地，TYPE_CLOUD 表示云端，TYPE_MIX 表示混合
RESULT_TYPE	设置返回结果格式。比如 json 表示 json 格式
LANGUAGE	设置语言。zh_cn 表示中文，en_us 表示英文
ACCENT	设置方言。mandarin 表示普通话，cantonese 表示粤语，henanese 表示河南话
VAD_BOS	设置静音超时时间，即用户多长时间不说话就当作超时处理
VAD_EOS	设置静音检测时间，即用户停止说话多长时间内就认为不再输入，自动停止录音
ASR_PTT	设置标点符号。0 表示返回结果无标点，1 表示返回结果有标点
AUDIO_FORMAT	设置音频的保存格式
ASR_AUDIO_PATH	设置音频的保存路径
AUDIO_SOURCE	设置音频的来源。-1 表示音频流，与 writeAudio 配合使用；-2 表示外部文件，同时设置 ASR_SOURCE_PATH 指定文件路径
ASR_SOURCE_PATH	设置外部音频文件的路径

- startListening：开始监听语音。参数为 RecognizerListener 对象，该对象需要重写以下方法。
  - onBeginOfSpeech：内部录音机已准备好，用户可以开始语音输入。
  - onError：错误码 10118（您没有说话），可能是录音机权限被禁，需要提示用户打开应用的录音权限。
  - onEndOfSpeech：检测到了语音的尾端点，已经进入识别过程，不再接收语音输入。
  - onResult：识别结束，返回结果串。
  - onVolumeChanged：语音输入过程中的音量大小变化。
  - onEvent：事件处理，一般是业务出错等异常。
- stopListening：结束监听语音。
- writeAudio：把指定的音频流作为语音输入。



- cancel：取消监听。
- destroy：回收语音识别对象。

语音识别的演示效果如图 15-44 和图 15-45 所示。其中，图 15-44 所示为点击“开始”按钮后，测试 App 正在倾听用户朗读时的界面；朗读完毕，语音 SDK 对音频流进行识别处理，并把语音识别后的文本内容显示在页面上，如图 15-45 所示。



图 15-44　测试 App 正在倾听用户说话

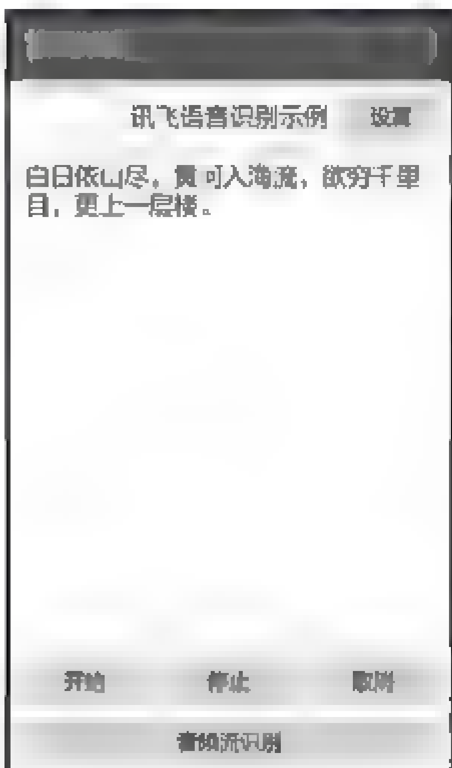


图 15-45　测试 App 显示语音识别的文本

15.4.2　语音合成

语音合成和语音识别功能在同 一个开发包中，只需 一次集成，无须重复。

讯飞 SDK 的语音合成功能主要通过 SpeechSynthesizer 类实现，有以下常用方法。

- createSynthesizer：创建语音合成对象。
- setParameter：设置语音合成的参数。语音合成的参数说明见表 15-6。

表15-6　语音合成的参数说明

SpeechConstant 类的合成参数	说明
ENGINE_TYPE	设置合成引擎。TYPE_LOCAL 表示本地，TYPE_CLOUD 表示云端，TYPE_MIX 表示混合
VOICE_NAME	设置朗读者。默认 xiaoyan（女青年，普通话）
SPEED	设置朗读的语速，取值 0～100
PITCH	设置朗读的音调，取值 0～100
VOLUME	设置朗读的音量，取值 0～100
STREAM_TYPE	设置音频流类型。默认是 3，表示音乐
KEY_REQUEST_FOCUS	设置是否在播放合成音频时打断音乐播放，默认为 true
AUDIO_FORMAT	设置音频的保存格式
TTS_AUDIO_PATH	设置音频的保存路径

- startSpeaking：开始语音朗读。参数为 SynthesizerListener 对象，该对象需重写以下方法。
  - onSpeakBegin：朗读开始。



  - ➢ onSpeakPaused：朗读暂停。
  - ➢ onSpeakResumed：朗读恢复。
  - ➢ onBufferProgress：合成进度变化。
  - ➢ onSpeakProgress：朗读进度变化。
  - ➢ onCompleted：朗读完成。
  - ➢ onEvent：事件处理，一般是业务出错等异常。
- synthesizeToUri：只保存音频不进行播放，调用该接口就不能调用 startSpeaking。第一个参数是要合成的文本，第二个参数是要保存的音频全路径，第三个参数是 SynthesizerListener 回调接口。
- pauseSpeaking：暂停朗读。
- resumeSpeaking：恢复朗读。
- stopSpeaking：停止朗读。
- destroy：回收语音合成对象。

语音合成的演示效果如图 15-46 和图 15-47 所示。其中，图 15-46 所示为选择发音人的对话框，可以看到讯飞语音提供了中文、英文以及汉语的常见方言，还是很丰富动听的；接着点击“开始合成”按钮，语音 SDK 对测试诗歌的文本进行语音合成，并播放合成后的音频流，如图 15-47 所示。

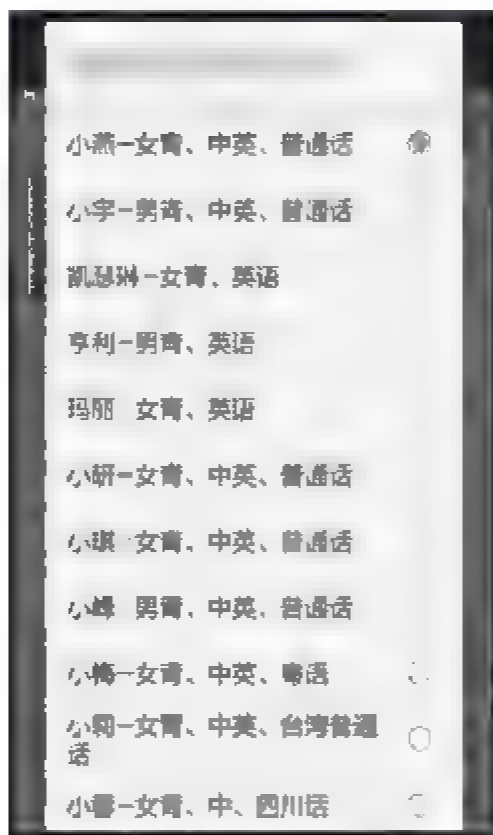


图 15-46　选择发音人的弹窗页面

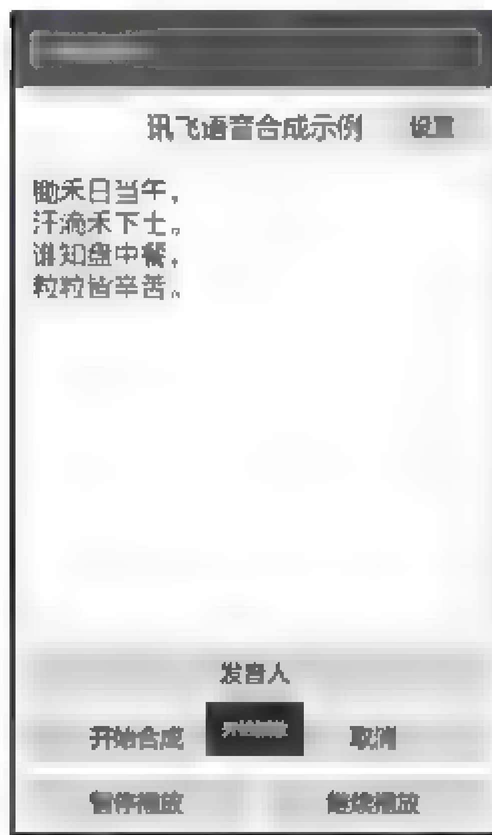


图 15-47　正在播放合成的语音

15.5　实战项目：仿滴滴打车

这几年分享经济如火如荼，从阿姨外卖到滴滴打车，都离不开新技术、新思想的实践。特别是打车 App，大家或多或少都用过，看起来很方便，可是背后的技术支持着实不简单。读者是否想过自己实现一个类似的打车 App 呢？现在就让我们一步一个脚印，开始着手吧！就算没法做出真正可用的打车 App，也要鼓捣一个演示用的“嗒嗒打车”。

15.5.1 设计思路

滴滴打车的用户界面主要是一幅地图配上相关的打车信息，打车的具体流程不外乎是：用户开始打车→司机接单→司机开到出发地，用户上车→司机开到目的地，用户下车→用户付款行程结束。这里为了突出本章的知识点，依然是化繁为简，把不怎么相关的控件元素去掉，形成山寨后的效果，如图 15-48 和图 15-49 所示。其中，图 15-48 所示为打车 App 的初始页面，图 15-49 所示为行程结束后的评价页面。

惯例还是“大家来找茬”，看看这个“嗒嗒打车”用到了本章的哪些知识点，想必读者早已轻车熟路全部找出来了。

（1）地图 SDK：主界面上都是地图，还得通过地图显示用户当前位置和快车的行车路线。

（2）语音 SDK：每当遇到一个需要提醒司机、用户的事件或路况信息，比如“快车已经到达”“前方五十米右转”等，都会响起一阵悦耳的女声播报。

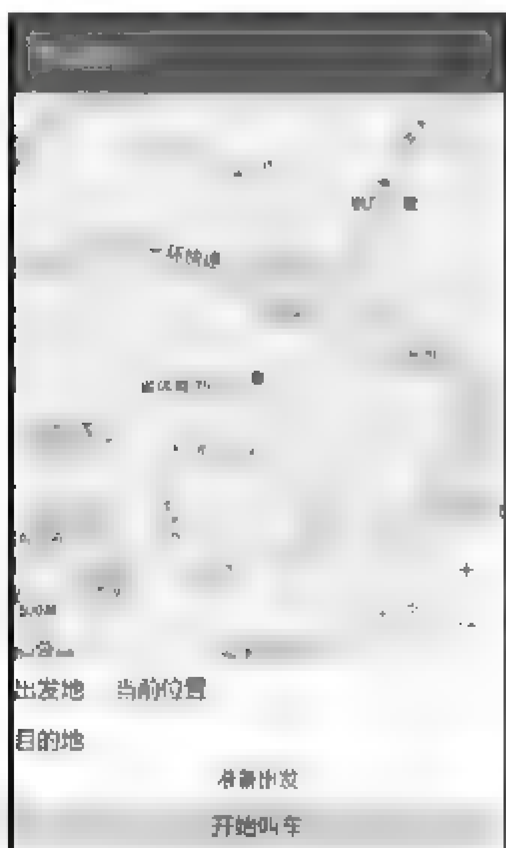


图 15-48 打车 App 的初始页面



图 15-49 行程结束后的评价页面

（3）支付 SDK：行程结束，用户通过支付宝或微信支付，把打车费付款给打车平台，由打车平台向司机分成。

（4）分享 SDK：体验到快车的方便快捷，小伙伴们想不想分享给好友呢？分享成功有红包哦。

真实的打车 App 还会用到更多第三方开发包，比如消息推送 SDK、统计分析 SDK 等。不过，实战项目的“嗒嗒打车”仅用于学习演示，能熟练运用上面 4 个 SDK 已经足够了。

15.5.2 小知识：评分条 RatingBar

在服务行业中，商家信誉是一个很重要的指标，信誉好的商户，生意自然越来越好。如何评价一个商户的信誉等级呢？这依赖于消费者每次光顾后的星级评价。无论是在淘宝购物，还是使用滴滴打车，订单结束了都会提示用户进行评价，此时用到的评价控件为评分条 RatingBar。



RatingBar 其实是拖动条 SeekBar 的升级版，不同之处在于把进度标记换成了五角星。RatingBar 除了拥有 SeekBar 的所有方法，还新增了 5 个与评分相关的方法，新增的方法与属性说明见表 15-7。

表15-7　RatingBar的新增方法与属性说明

XML 的新增属性	RatingBar 的新增方法	说明
isIndicator	setIsIndicator	是否作为指示器。如果是指示器，就不可通过触摸修改评级
numStars	setNumStars	设置星星的个数
rating	setRating	设置初始评价等级
stepSize	setStepSize	设置每次增减的大小。默认为总数的十分之一，比如星星总数为 5，默认值为 0.5
无	setOnRatingBarChangeListener	设置评分监听器。 需实现接口 OnRatingBarChangeListener 的 onRatingChanged 方法

另外，RatingBar 提供了 3 种星星样式，用于不同业务场景时的评级展示。评分条的样式说明见表 15-8。

表15-8　评分条的样式说明

评分条 style 属性的风格	星星的规格大小	默认能否触摸改变评级
?android:attr/ratingBarStyle	大，默认值	能
?android:attr/ratingBarStyleIndicator	中	不能
?android:attr/ratingBarStyleSmall	小	不能

尽管 RatingBar 提供了 3 种星星样式，不过是换汤不换药，评分条的星星外观仍然不尽如人意。如果想定制星星的颜色与大小，就得自定义一个层次图形描述文件，然后把 RatingBar 的 progressDrawable 属性设置为该层次图形。下面是自定义层次图形 XML 文件定义代码：

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+android:id/background"
        android:drawable="@drawable/star_background">
    </item>
    <item
        android:id="@+android:id/secondaryProgress"
        android:drawable="@drawable/star_background">
    </item>
    <item
        android:id="@+android:id/progress"
        android:drawable="@drawable/star_foreground">
    </item>
</layer-list>
```

下面是使用 RatingBar 的代码：

```
public class RatingBarActivity extends AppCompatActivity implements
        OnCheckedChangeListener, OnRatingBarChangeListener {
    private CheckBox ck_whole;
    private RatingBar rb_score;
    private TextView tv_rating;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rating_bar);
        ck_whole = (CheckBox) findViewById(R.id.ck_whole);
        rb_score = (RatingBar) findViewById(R.id.rb_score);
        tv_rating = (TextView) findViewById(R.id.tv_rating);
        ck_whole.setOnCheckedChangeListener(this);
        rb_score.setOnRatingBarChangeListener(this);
    }

    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (buttonView.getId() == R.id.ck_whole) {
            rb_score.setStepSize(ck_whole.isChecked()?1:rb_score.getNumStars()/10.0f);
        }
    }

    @Override
    public void onRatingChanged(RatingBar ratingBar, float rating, boolean fromUser) {
        String desc = String.format("当前选中的是%s 颗星", CacheUtil.formatDecimal(rating,1));
        tv_rating.setText(desc);
    }
}
```

评分条的演示效果如图 15-50 和图 15-51 所示。图 15-50 所示为选择两颗星时的效果图。图 15-51 所示为选择 3 颗半星时的效果图。

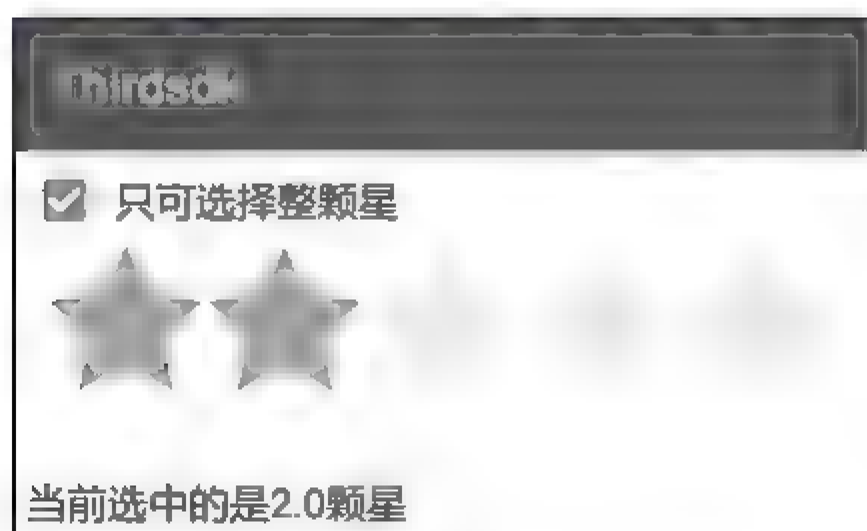


图 15-50 选择两颗星时的效果图



图 15-51 选择 3 颗半星时的效果图

15.5.3 代码示例

编码与测试方面需要注意以下 4 点：

（1）在 libs 目录与 src/main/jniLibs 目录下正确放置相关的 SDK 文件。

（2）AndroidManifest.xml注意声明相关权限，并注册地图APPKEY和相应的activity和service。

（3）注意地图服务与语音服务的初始化操作。

（4）使用真机测试体验效果更佳。

其余编码没什么难点了，赶紧把“嗒嗒打车”安装到手机上，试着完整运行一遍，看看是什么感觉。或者先看笔者这里的测试效果图，一点都不难，你也可以的。一开始打开测试 App，填写出发地与目的地，然后点击“开始叫车”按钮，App 发布打车请求，并语音播报“等待司机接单”，如图 15-52 所示。司机接单后，App 语音播报“司机马上过来”，因为截图体现不了声音，所以页面下方另外加了一排文字显示语音播报的内容，如图 15-53 所示。

快车到达用户位置后，小车图标与用户圆点重合，同时 App 语音播报“快车已经到达，请上车”，如图 15-54 所示。然后用户上车，司机一路开向目的地，App 语音播报“已经到达目的地，欢迎下次再来乘车”，同时下方按钮的文字变为“支付车费”，如图 15-55 所示。

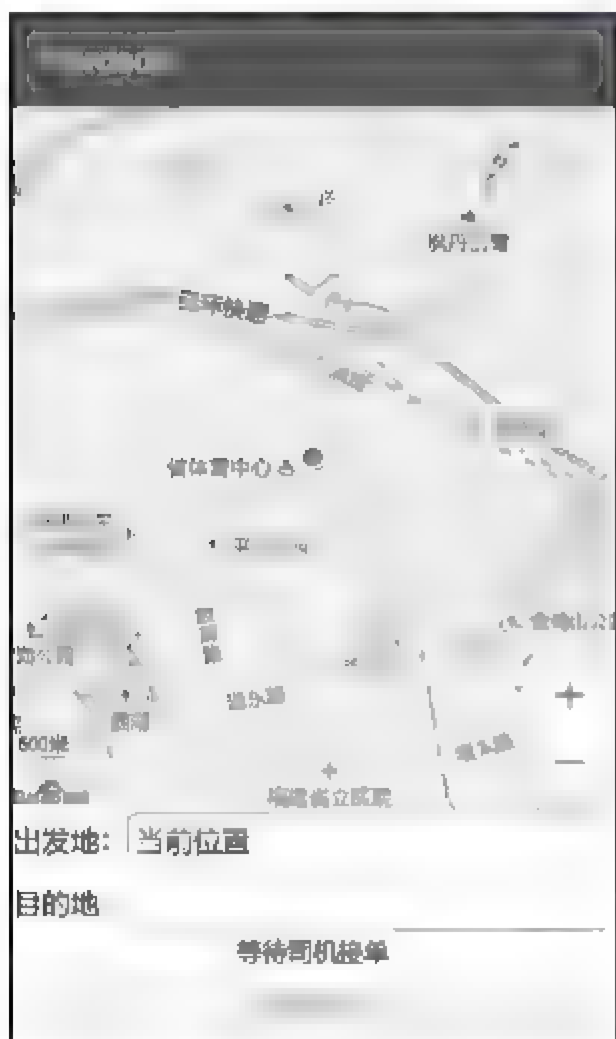


图 15-52　等待司机接单时的界面

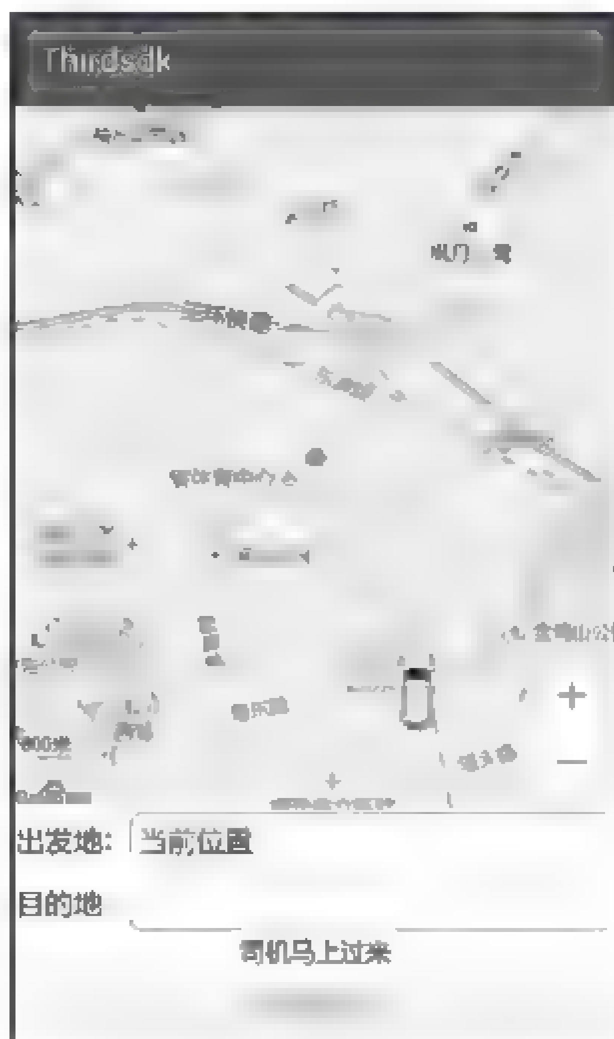


图 15-53　司机马上过来的界面

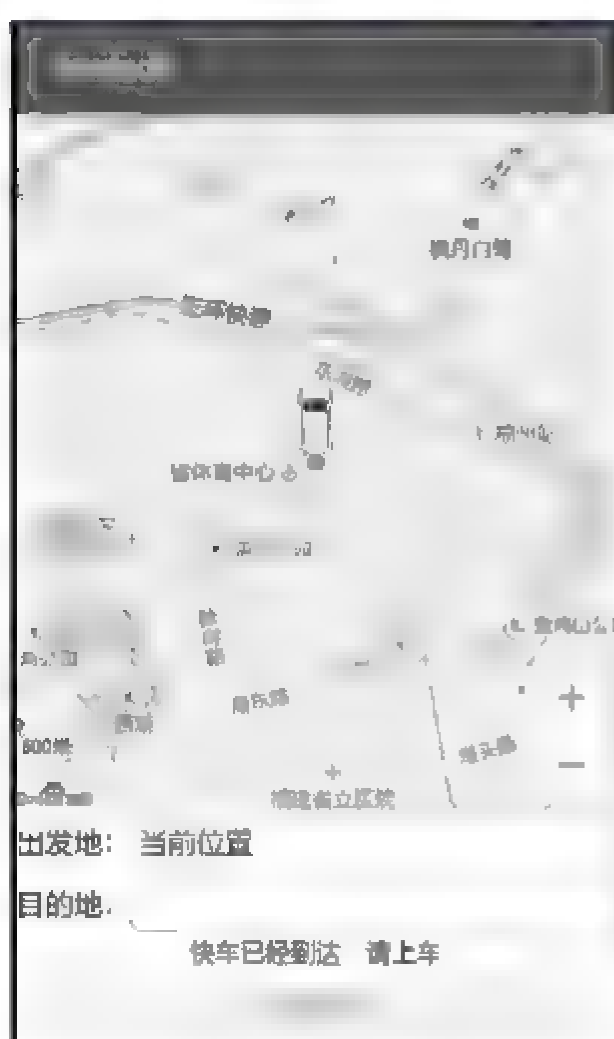


图 15-54　快车过来接客时的界面

用户点击“支付车费”按钮，页面下方弹出付款对话框，如图 15-56 所示。确认付款信息正确无误后，点击“确认付款”按钮完成支付操作，然后跳到评价页面，用户可在此给快车服务打分，也可将打车信息分享给好友，如图 15-57 所示。

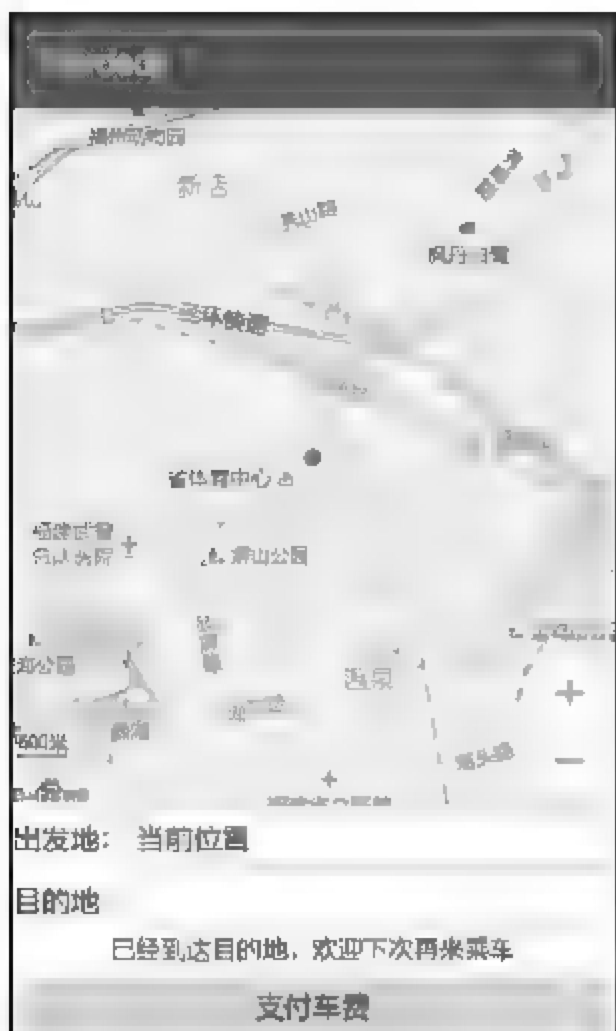


图 15-55　打车行程结束时的界面

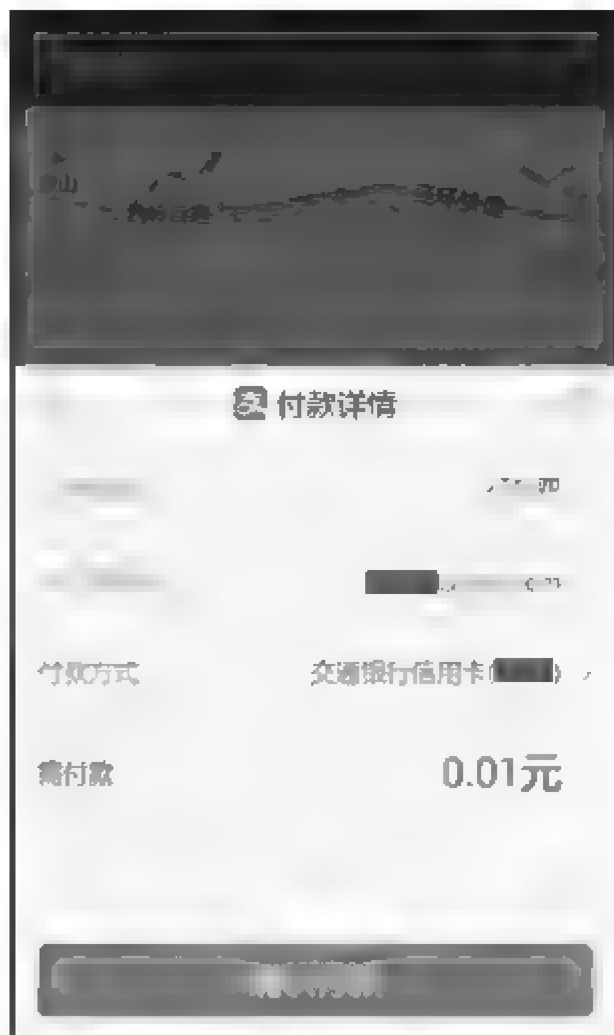


图 15-56　支付车费的付款对话框



图 15-57　评价完成时的页面

15.6 小　结

本章主要介绍了 App 开发用到的常见第三方开发包，包括地图 SDK（查看签名信息、百度地图、高德地图）、分享 SDK（QQ 分享、微信分享）、支付 SDK（支付宝、微信支付）、语音 SDK（语音识别、语音合成）。最后设计了一个实战项目“仿滴滴打车”，在该项目的 App 编码中采用了本章讲述的 4 种开发包的代表技术。另外，介绍了如何使用评分条。

通过本章的学习，读者应该能够掌握以下 4 种开发技能：

（1）学会使用地图 SDK 进行定位、搜索、测量等操作。
（2）学会使用分享 SDK 把消息内容分享到 QQ 与微信。
（3）初步了解支付的交易流程，并学会使用支付 SDK 演示支付缴费功能。
（4）学会使用语音 SDK 完成语音的识别和语音的合成。



性能优化

本章介绍 App 开发常见的性能优化技术，主要包括通过优化布局文件实现页面风格的统一、通过检测手段和预防措施处理内存泄漏的问题、运用线程池技术对线程资源进行有效管理、通过监测当前电量与屏幕事件开启省电模式，最后结合本章所学的知识演示一个实战项目“图片缓存框架”的设计与实现。

16.1 布局文件优化

Android 的页面布局千变万化，但对某个具体的 App 来说，往往要求有统一的风格，比如统一的导航栏、统一的竖屏布局与横屏布局、统一的窗口主题等，这种统一风格就像学生的校服和白领的制服。本节介绍风格统一的几种方式，包括增加公共布局减少重复布局、使用占位视图自适应调整屏幕布局、自定义窗口主题等内容。

16.1.1 减少重复布局

第 7 章介绍工具栏 Toolbar 的时候提到在布局文件中加入该节点实现顶部导航栏效果。由于 App 内部存在多个活动页面，为了确保所有页面的风格统一，因此必须给每个页面的布局文件添加 Toolbar。如此一来，这些 XML 文件几乎包含一模一样的 Toolbar 布局，不但造成重复布局，而且不易扩展，因为每往导航栏上增加一个新控件，都得把涉及的 XML 文件统统修改过去。

这种重复的导航栏布局，若能参照代码中的公共函数抽出来形成单独的公共布局文件，由各个页面布局文件分别引用，岂不妙哉？Android 确实提供了对应的途径，只要在页面布局中使用 include 标签声明公共布局，即可实现在该页面中导入公共布局内容，功能类似于 Java 的 import 或 C/C++的 include 关键字。include 标签适用于在多个布局文件中导入相同的 XML 布局片段，比如相同的标题栏、相同的广告栏、相同的进度栏等。include 标签的用法很简单，只需一行配置即可完成公共布局引用，如下面的代码表示引用了一个名为 common_title.xml 的公共布局文件：

```
<include layout="@layout/common_title" />
```

公共布局文件的根节点可以是 LinearLayout、RelativeLayout 等布局节点，不过外部的页面布局文件往往已经有了相同的布局节点，这时子布局的根节点就变成冗余的了，但是布局文件必须有根布局节点，不能把控件作为根节点。为了解决根布局冗余的问题，Android 提供了 merge 标签进行布局优化，即把 merge 标签作为公共布局文件的根节点。merge 标签代替了 LinearLayout、RelativeLayout 等原根节点的位置，也就是告诉编译器：我只是一个占位的合并标签，不需要对我做布局处理。这样，App 在渲染界面时只是原样导入 merge 标签下的视图内容，不做根布局尺寸的计算和调整，从而提高了 UI 的加载效率。

为了更好理解 include 与 merge 标签的用法，接下来举一个公共布局文件的例子：

```
<merge xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto" >

    <android.support.v7.widget.Toolbar
        android:id="@+id/tl_head"
        android:layout_width="match_parent"
```

```
            android:layout_height="50dp"
            android:background="@color/blue_light"
            app:navigationIcon="@drawable/ic_back" >

            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content" >

                <TextView
                    android:id="@+id/tv_title"
                    android:layout_width="wrap_content"
                    android:layout_height="match_parent"
                    android:layout_centerInParent="true"
                    android:paddingRight="50dp"
                    android:textColor="@color/black"
                    android:textSize="20sp" />

                <ImageView
                    android:id="@+id/iv_share"
                    android:layout_width="wrap_content"
                    android:layout_height="match_parent"
                    android:layout_alignParentRight="true"
                    android:src="@drawable/ic_share"
                    android:scaleType="fitCenter" />
            </RelativeLayout>
        </android.support.v7.widget.Toolbar>
</merge>
```

处理公共布局必然要有对应的公共页面代码，为此我们声明一个名为 BaseActivity 的活动基类，该基类默认处理公共布局中的控件操作，具体的活动页面由 BaseActivity 派生而来。活动基类的示例代码如下：

```
public class BaseActivity extends AppCompatActivity {
    @Override
    protected void onResume() {
        super.onResume();
        Toolbar tl_head = (Toolbar) findViewById(R.id.tl_head);
        setSupportActionBar(tl_head);
        tl_head.setNavigationOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View view) {
                finish();
            }
        });
```

```
                findViewById(R.id.iv_share).setOnClickListener(new OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        Toast.makeText(BaseActivity.this, "请先实现分享功能", Toast.LENGTH_LONG).
show();
                    }
                });
            }

        protected void setTitle(String title) {
            TextView tv_title = (TextView) findViewById(R.id.tv_title);
            tv_title.setText(title);
        }
    }
```

最后给出两个实际页面的布局，分别使用 include 标签导入公共布局 common_title，然后在代码中分别从 BaseActivity 派生两个具体的页面类。其中一个页面的代码举例如下：

```
    public class IncludeOneActivity extends BaseActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_include_one);
            setTitle("时事频道");
        }
    }
```

运行后的公共导航栏效果如图 16-1 和图 16-2 所示。图 16-1 所示为第一个时事频道的界面。图 16-2 所示为第二个体育频道的界面。

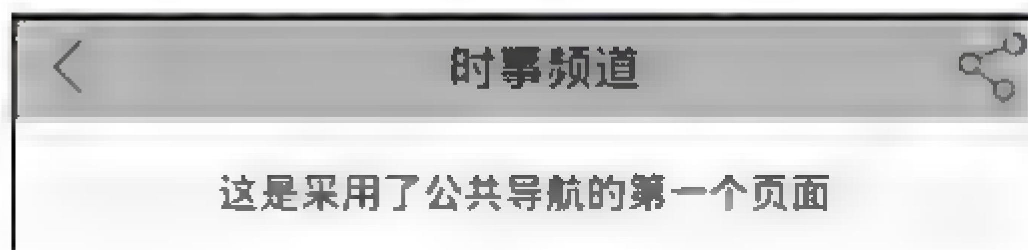


图 16-1 时事频道页面



图 16-2 体育频道页面

16.1.2 自适应调整布局

在页面上根据条件展示不同的视图常常需要设置视图的可视属性。比如调用 setVisibility 方法设置可视属性，若需展示则将可视属性设置为 View.VISIBLE，若需隐藏则将可视属性设置为 View.GONE。然而 gone 的视图只是看不到罢了，在界面渲染时还是会被加载。要想事先不加载视图，在条件匹配时才加载，就可以使用标签 ViewStub。

占位视图 ViewStub 类似一个简单的 View，但其内部布局由属性 layout 指定。在 App 加载页面时，ViewStub 并不显示布局内容，只有在代码中调用 ViewStub 对象的 inflate 方法时，

layout 指定的布局才会展示出来。基于以上处理逻辑，ViewStub 在提高布局性能上有以下两个特点：

（1）ViewStub 在加载时只占用大约一个 View 的内存，不占用 layout 整个布局需要的内存。

（2）ViewStub 一旦调用 inflate 方法，就立即显示所包含的页面内容。如果还想再次隐藏或显示布局，就要通过 setVisibility 方法实现。

举一个 ViewStub 实际运用的例子，手机在竖屏和横屏之间切换时，有时希望显示不同的布局，比如竖屏显示列表、横屏显示网格。如此一来，在页面布局中预留两个 ViewStub 节点，一个给 ListView 占位，另一个给 GridView 占位，具体的布局内容如下：

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <include layout="@layout/common_title" />

    <ViewStub
        android:id="@+id/vs_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout="@layout/viewstub_list" />

    <ViewStub
        android:id="@+id/vs_grid"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout="@layout/viewstub_grid" />
</LinearLayout>
```

相对应的，页面代码增加对横竖屏的方向判断，如果当前为竖屏，就令占位视图显示列表布局；如果当前为横屏，就令占位视图显示网格布局。页面代码举例如下：

```
public class ScreenSuitableActivity extends BaseActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_screen_suitable);
        setTitle("自适应布局演示页面");
        Configuration config = getResources().getConfiguration();
        if(config.orientation == Configuration.ORIENTATION_PORTRAIT){
            showList();
        } else {
            showGrid();
```

```
            }
        }

        private void showList() {
            ViewStub vs_list = (ViewStub) findViewById(R.id.vs_list);
            vs_list.inflate();
            ListView lv_hello = (ListView) findViewById(R.id.lv_hello);
            PlanetAdapter adapter = new PlanetAdapter(this, R.layout.item_list,
                    Planet.getDefaultList(), Color.WHITE);
            lv_hello.setAdapter(adapter);
        }

        private void showGrid() {
            ViewStub vs_grid = (ViewStub) findViewById(R.id.vs_grid);
            vs_grid.inflate();
            GridView gv_hello = (GridView) findViewById(R.id.gv_hello);
            PlanetAdapter adapter = new PlanetAdapter(this, R.layout.item_grid,
                    Planet.getDefaultList(), Color.WHITE);
            gv_hello.setAdapter(adapter);
        }
    }
```

上述自适应布局的演示效果如图 16-3 和图 16-4 所示。图 16-3 所示为展示列表的竖屏界面。图 16-4 所示为展示网格的横屏界面。

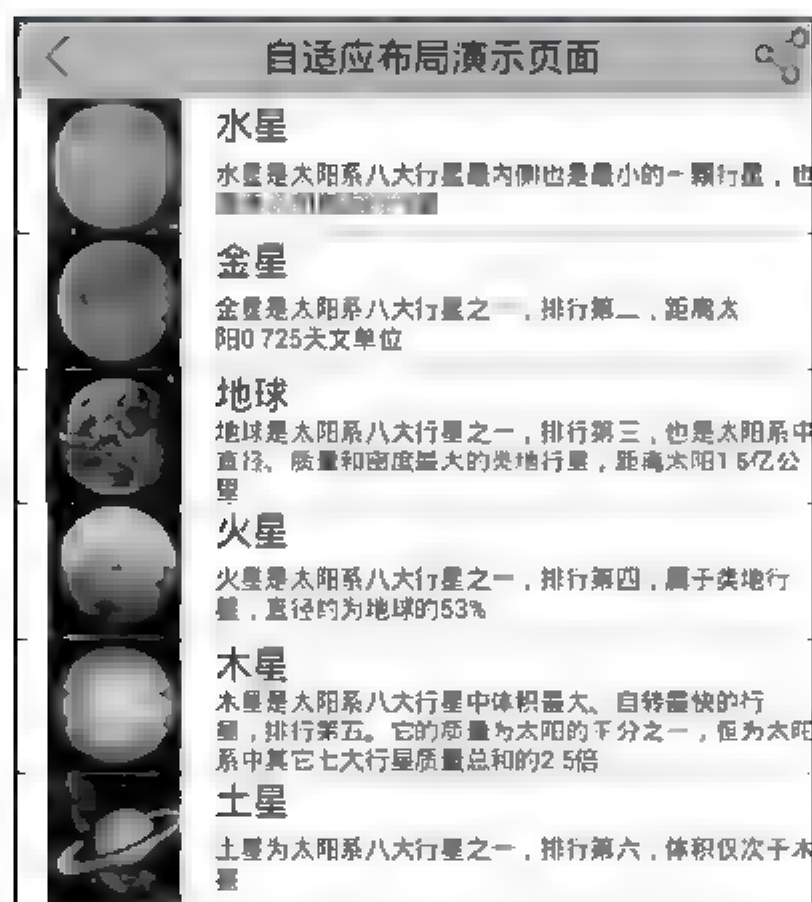


图 16-3　占位视图展示竖屏列表

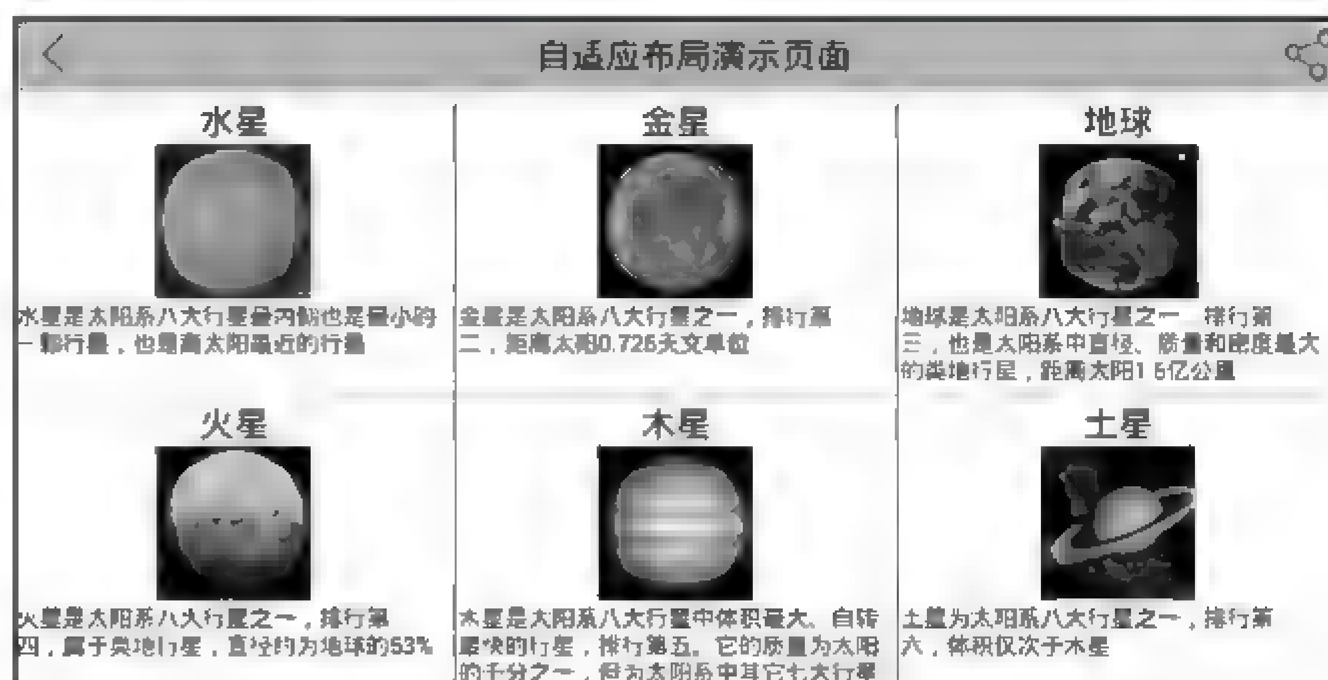


图 16-4　占位视图展示横屏网格

16.1.3　自定义窗口主题

使用 Android Studio 创建一个新模块，默认的 App 主题为系统自带的 Theme.AppCompat.Light.DarkActionBar，即浅灰背景加深色导航栏。如果大家都用默认主题，

App 势必变得千篇一律、毫无特色。要想让自己的 App 吸引眼球，首先得打造非同一般的主题，比如粉红的小女人风格、草绿的小清新风格、天蓝的闷骚男风格等。

自定义主题的配置可在 res/values/styles.xml 中定义，配置方式同一般视图的 style 风格配置，不同的是如何应用自定义主题。一般视图可在布局文件的节点中使用 style 属性设置风格，对于视窗则可通过以下途径设置主题：

（1）修改 AndroidManifest.xml，往 application 节点增加 android:theme 属性，表示对该 App 的所有页面设置指定的主题；或者往 activity 节点增加 android:theme 属性，表示对指定的活动页面单独设置主题。

（2）打开 Activity 代码，在 setContentView 方法之前调用方法 setTheme(R.style.***)完成该页面的主题设置。

（3）如果是自定义对话框，就在 Dialog 的构造函数中传入指定主题的资源编号。

下面介绍窗口主题经常需要自定义的属性。

- android:gravity：窗口内部的对齐方式。
- android:background：窗口内部的背景。
- android:windowBackground：整个窗口的背景，包括边框与内部。
- android:windowFrame：窗口框架图像。注意该属性并不只是边框区域，还包括内部窗口，所以如果 windowFrame 设置为不透明的图像，那么内部窗口将只显示这幅不透明的图像。
- android:windowNoTitle：窗口是否不要默认的标题栏，即是否展示 ActionBar。
- android:windowFullscreen：窗口是否全屏。
- android:windowIsTranslucent：窗口是否半透明。
- android:windowIsFloating：窗口是否悬浮。
- android:windowAnimationStyle：窗口切换动画的样式。
- android:windowEnterAnimation：进入窗口的动画。
- android:windowExitAnimation：退出窗口的动画。

在以上属性中，与背景设置有关的 3 个属性容易混淆，分别是 android:windowFrame、android:windowBackground 和 android:background。下面测试一下这 3 个属性对应的视窗界面，看看究竟是什么模样。

首先设定页面背景是绿色，接着将 android:windowBackground 设置为半透明红色，效果如图 16-5 所示。此时对话框外围变为深黄绿色，即窗口对外半透明，使得页面背景与窗口背景混合在一起。



图 16-5　windowBackground 设置为半透明红色的效果

然后将 android:background 设置为半透明红色，效果如图 16-6 所示。此时对话框外围变为红色，四周边框为深绿色，表示窗口内部对外不透明，但窗口边框对外透明。

最后将 android:windowFrame 设置为半透明红色，效果如图 16-7 所示。此时对话框内部蒙上半透明红色，四周边框变为黄绿色，说明窗口内部对外不透明但对内半透明，窗口边框对外半透明。

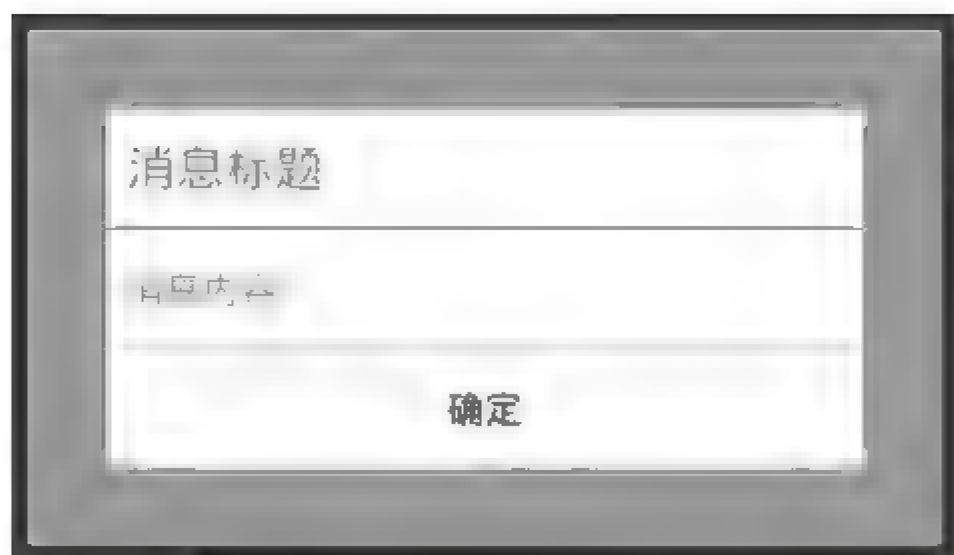


图 16-6 background 设置为半透明红色的效果

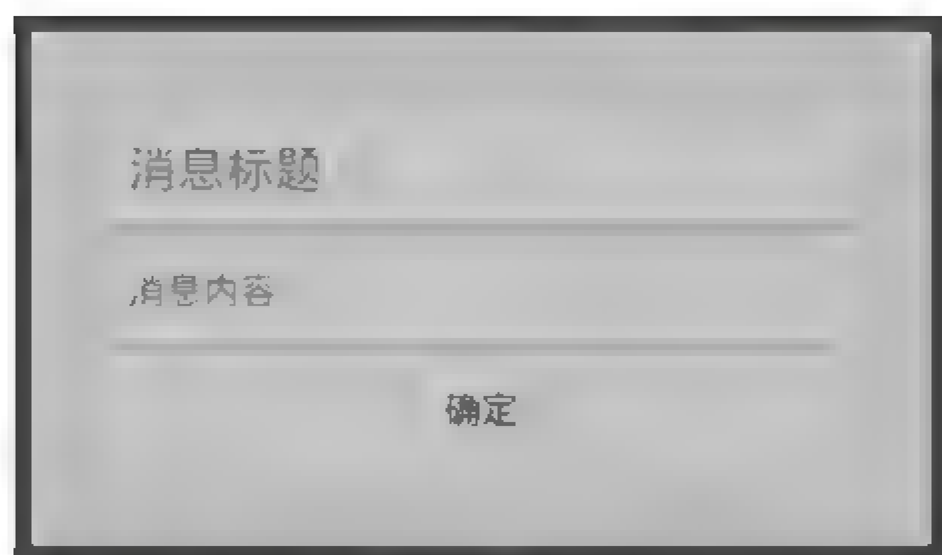


图 16-7 windowFrame 设置为半透明红色的效果

16.2 内存泄漏处理

内存泄漏指的是程序运行时未能正确回收部分内存，导致这些内存既不能被自身使用，又不能被其他程序使用，从而变成垃圾内存。一旦内存泄漏无法得到控制，该程序占用的内存就越来越大，最终只能强行结束，否则会导致系统死机。本节首先介绍 Android 开发如何检测内存泄漏，然后详细阐述各种场景下的内存泄漏预防措施。

16.2.1 内存泄漏的检测

C/C++存在指针的概念，每当程序需要处理数据时，便从内存中开辟一块区域，并把该区域的首地址赋值给一个指针，这样程序才能够操作该指针指向的内存。因为 C/C++设计上的原因，手工分配的内存也要手工释放，如果没有及时释放内存就会产生内存泄漏。Java 设计之初已经实现了多数情况的内存自动回收，不过在 Android 开发中，内存回收机制并不总会奏效。情况一是调用了非 Java 接口，比如调用了 JNI 接口，JNI 代码中由 C/C++分配的内存就要手工回收；情况二是调用了外部服务，使用完毕就得手工通知外部服务回收；情况三是异步处理，实时的内存回收机制显然等不了耗时较久的异步处理任务。

要对内存泄漏问题进行优化，首先得检测 App 是否发生内存泄漏。正常情况下，一个 App 占用的内存有一个峰值，达到这个峰值后，只要退出 App 页面，占用的内存大小就会降下来。但是如果产生内存泄漏，这个 App 占用的内存大小是没有峰值的，随着页面的重复打开或时间的不断流逝，该 App 消耗的内存越变越大，这便表示出现了内存泄漏状况。因此，只要能够监控 App 的运行内存变化情况，即可间接判断这个 App 是否发生内存泄漏。

Android Studio 自带了简单的内存检测工具，使用 Android Studio 运行测试应用时，在主界面左下角的 logcat 窗口左侧从上往下数第 3 个按钮为 System Information，单击该按钮会在右边弹出菜单窗口，如图 16-8 所示。

菜单项中的 Memory Usage 表示查看内存用量。单击该菜单项，代码主窗口会显示测试设备上的内存使用情况，如图 16-9 所示。可以看到当前测试应用 performance 的内存消耗量为 8573KB，在该 App 上点击，打开一个包含多张图片的页面，再次单击 Memory Usage 查看内存用量，此时的内存统计结果如图 16-10 所示。

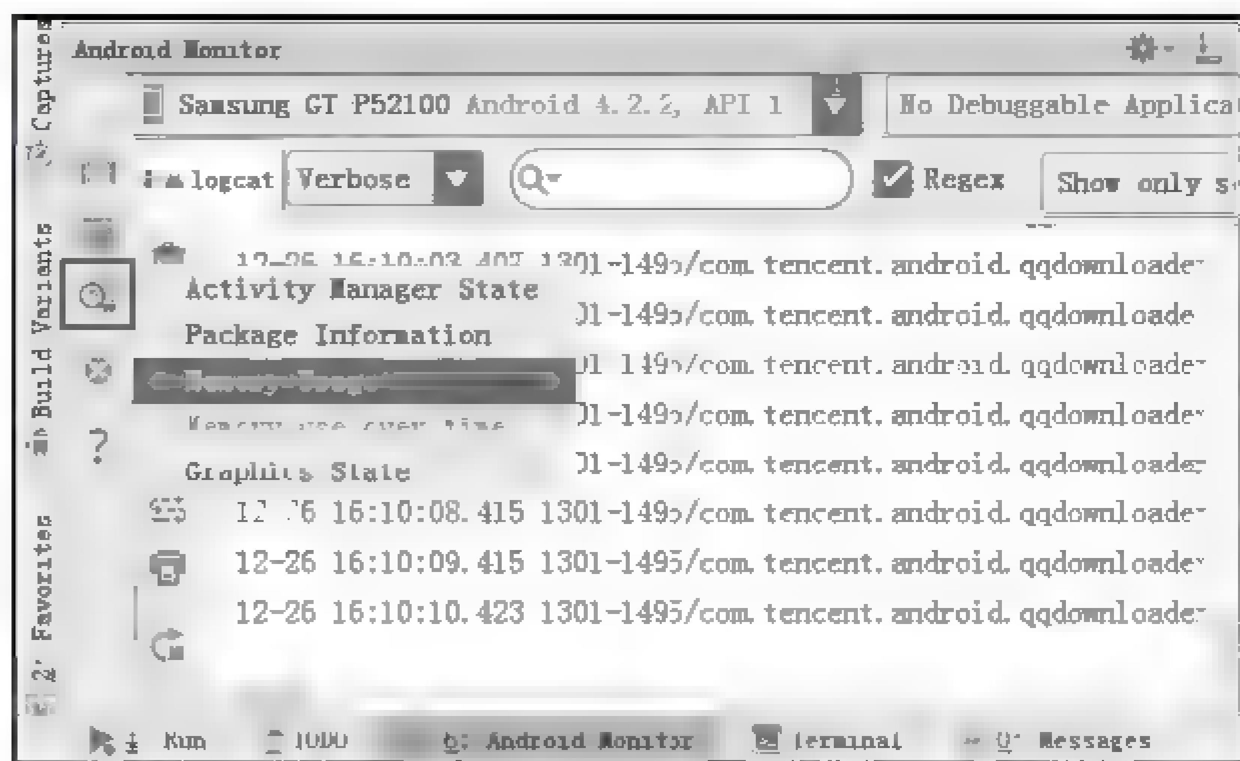


图 16-8 查看内存用量弹出的菜单

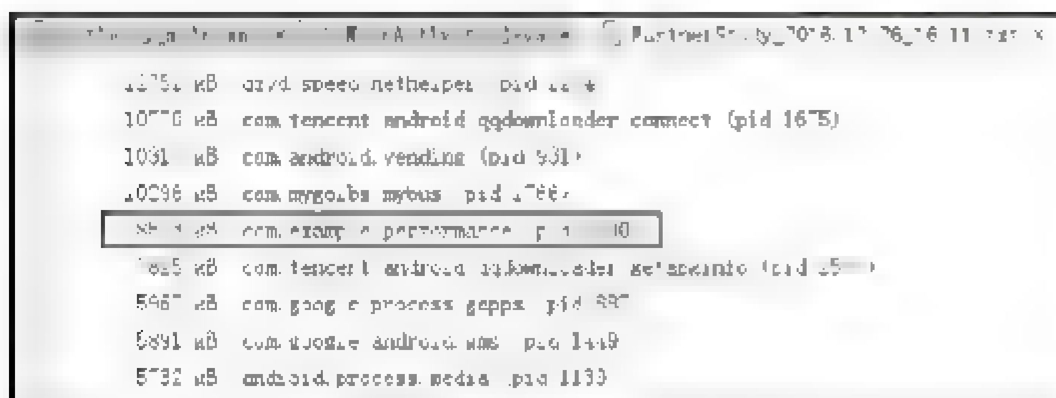


图 16-9 测试设备的初始内存使用状况

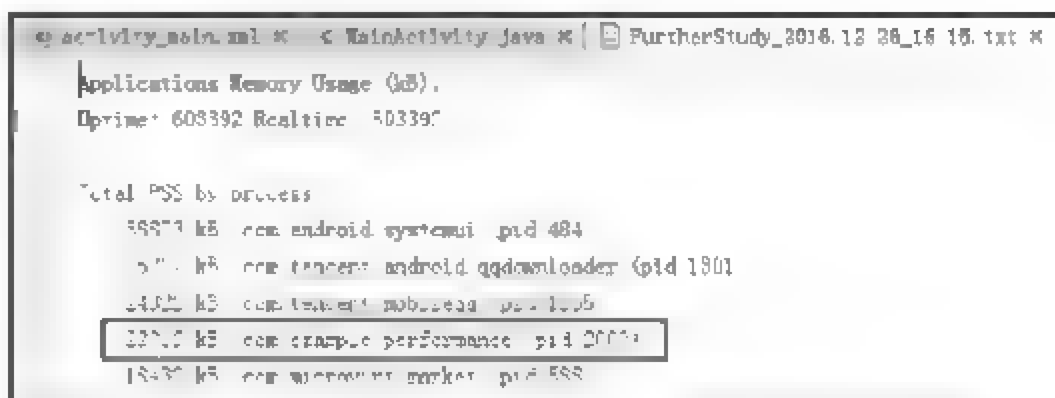


图 16-10 打开 App 页面后的内存使用状况

这时测试应用 performance 的内存消耗量达到了 22212KB，反复打开和关闭页面，然后重复内存查看操作。如果发现 App 占用的内存只增不减，那么毫无疑问发生了内存泄漏。

为进一步观察内存泄露现象，下面给出两个例子分别进行说明。

1. 未能移除定时的 Runnable 任务

前面各章有需要延时处理时，常常调用 Handler 对象的 postDelayed 方法，由该方法延迟一段时间后执行设定好的 Runnable 任务。若要实现动画效果，则循环执行若干次 postDelayed 方法。你可曾想过，这里蕴含着不小的内存泄漏风险，如果不谨慎对待，App 很可能多跑几次就挂了。

比如下面的代码每隔两秒打印一行日志，并在 onDestroy 页面退出时根据开关判断是否移除任务，完整代码如下：

```
public class RemoveTaskActivity extends AppCompatActivity implements OnClickListener {
    private boolean bRun = false;
    private CheckBox ck_remove;
    private TextView tv_remove;
    private Button btn_remove;
    private String mDesc = "";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_remove_task);
```

```
        ck_remove = (CheckBox) findViewById(R.id.ck_remove);
        tv_remove = (TextView) findViewById(R.id.tv_remove);
        btn_remove = (Button) findViewById(R.id.btn_remove);
        btn_remove.setOnClickListener(this);
        TextView tv_start = (TextView) findViewById(R.id.tv_start);
        tv_start.setText("页面打开时间为 ： "+Utils.getNowTime());
    }

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btn_remove) {
            if (bRun != true) {
                btn_remove.setText("取消定时任务");
                mHandler.post(mTask);
            } else {
                btn_remove.setText("开始定时任务");
                mHandler.removeCallbacks(mTask);
            }
            bRun = !bRun;
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (ck_remove.isChecked() == true) {
            mHandler.removeCallbacks(mTask);
        }
    }

    private Handler mHandler = new Handler();
    private Runnable mTask = new Runnable() {
        @Override
        public void run() {
            Intent intent = new Intent(TASK_EVENT);
            LocalBroadcastManager.getInstance(RemoveTaskActivity.this).sendBroadcast(intent);
            mHandler.postDelayed(this, 2000);
        }
    };

    @Override
    public void onStart() {
        super.onStart();
        taskReceiver = new TaskReceiver();
        IntentFilter filter = new IntentFilter(TASK_EVENT);
```

```
            LocalBroadcastManager.getInstance(this).registerReceiver(taskReceiver, filter);
        }

        @Override
        public void onStop() {
            LocalBroadcastManager.getInstance(this).unregisterReceiver(taskReceiver);
            super.onStop();
        }

        private String TASK_EVENT = "com.example.performance.task";
        private TaskReceiver taskReceiver;
        private class TaskReceiver extends BroadcastReceiver {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent != null) {
                    mDesc = String.format("%s%s 打印了一行测试日志\n", mDesc, Utils.getNowTime());
                    tv_remove.setText(mDesc);
                }
            }
        }
    }
```

首次进入该测试页面，点击“开始定时任务”按钮后，页面每隔两秒打印一行日志，如图 16-11 所示。然后不停止也不移除定时任务，直接退出该页面，按道理原测试页面上的内存都应该回收。不过接着重新进入测试页面，还没点击“开始定时任务”按钮，页面已经在兀自欢快地打印日志了，如图 16-12 所示，很明显上次退出页面时系统未能自动回收内存。

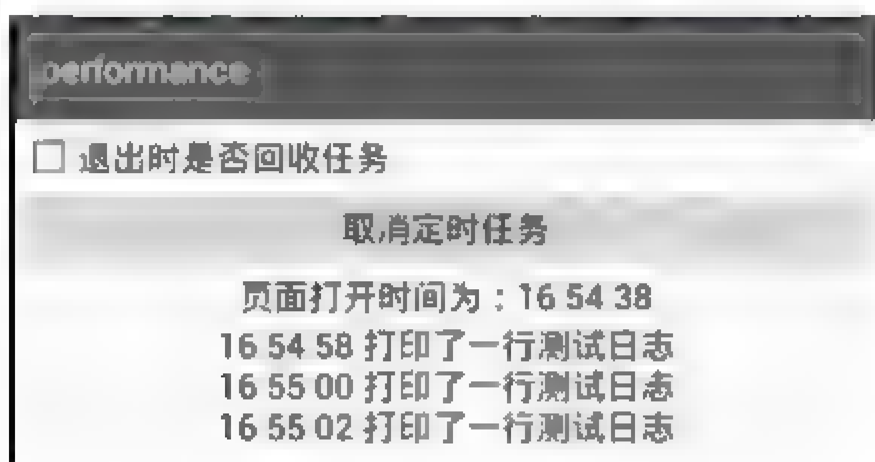


图 16-11　开始定时任务的测试页面

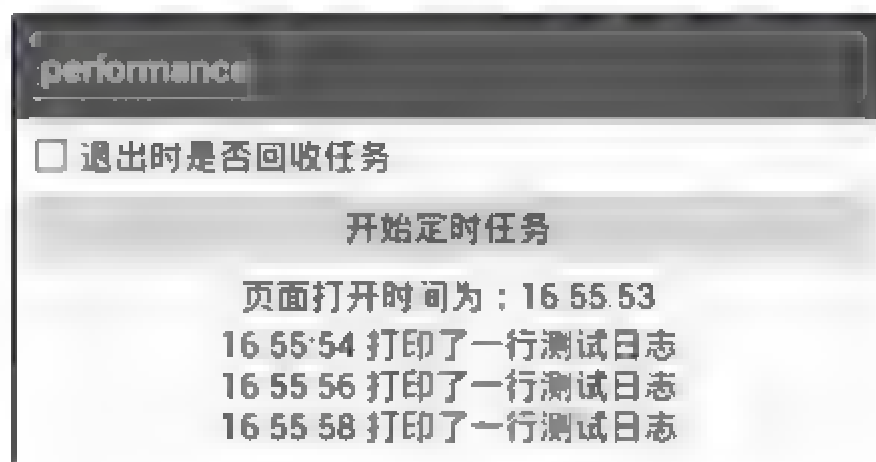


图 16-12　重新进入测试页面的情况

2. 未能注销系统的闹钟提醒服务

定时处理除了可以循环调用 postDelayed 方法外，还可以在系统的闹钟提醒服务中注册定时事件，并接收系统的闹钟广播进行定时处理。使用系统服务也需小心，因为系统的后台服务不知道 App 页面会在什么时候关闭，若放任自流，则又是一个内存泄漏的引爆点。

比如下面的代码利用闹钟提醒服务每隔 3 秒打印一行日志，并在 onDestroy 中根据开关判断是否注销服务，完整代码如下：

```
public class LogoutServiceActivity extends AppCompatActivity implements OnClickListener {
    private static final String TAG = "LogoutServiceActivity";
```

```
private boolean bRun = false;
private CheckBox ck_logout;
private Button btn_alarm;
private static TextView tv_alarm;
private PendingIntent pIntent;
private AlarmManager mAlarmManager;
private static String mDesc;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_logout_service);
    ck_logout = (CheckBox) findViewById(R.id.ck_logout);
    tv_alarm = (TextView) findViewById(R.id.tv_alarm);
    btn_alarm = (Button) findViewById(R.id.btn_alarm);
    btn_alarm.setOnClickListener(this);
    Intent intent = new Intent(SERVICE_EVENT);
    pIntent = PendingIntent.getBroadcast(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);
    mAlarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    mDesc = "";
    TextView tv_start = (TextView) findViewById(R.id.tv_start);
    tv_start.setText("页面打开时间为 ： "+Utils.getNowTime());
}

@Override
protected void onDestroy() {
    super.onDestroy();
    if (ck_logout.isChecked() == true) {
        mAlarmManager.cancel(pIntent);
    }
}

@Override
public void onClick(View v) {
    if (v.getId() == R.id.btn_alarm) {
        if (bRun != true) {
            mAlarmManager.setRepeating(AlarmManager.RTC_WAKEUP,
                    System.currentTimeMillis(), 3000, pIntent);
            mDesc = Utils.getNowTime() + " 设置闹钟";
            tv_alarm.setText(mDesc);
            btn_alarm.setText("取消闹钟");
        } else {
            mAlarmManager.cancel(pIntent);
```

```
                    btn_alarm.setText("设置闹钟");
                }
                bRun = !bRun;
            }
        }

        private String SERVICE_EVENT = "com.example.performance.service1";
        public static class ServiceReceiver extends BroadcastReceiver {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (intent != null) {
                    Log.d(TAG, "ServiceReceiver onReceive");
                    if (tv_alarm != null) {
                        mDesc = String.format("%s\n%s  闹钟时间到达", mDesc, Utils.getNowTime());
                        tv_alarm.setText(mDesc);
                    }
                }
            }
        }
    }
```

首次进入该测试页面，点击“设置闹钟”按钮后，页面每隔 3 秒打印一行日志，如图 16-13 所示。然后不取消也不注销闹钟服务，直接退出该页面，接着重新进入测试页面，还没点击设置按钮，页面却已经在不断刷新日志了，如图 16-14 所示，很遗憾上次的闹钟设置也产生了内存泄漏。

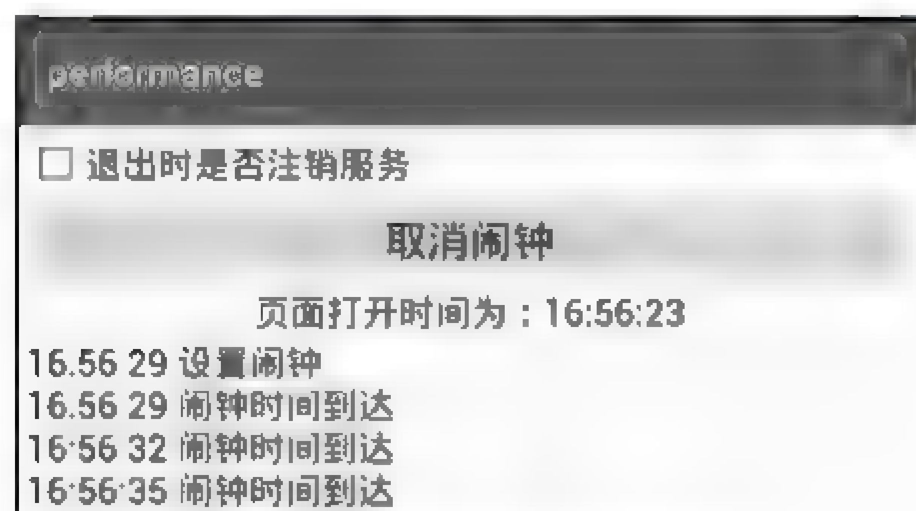


图 16-13　设置闹钟后的测试页面

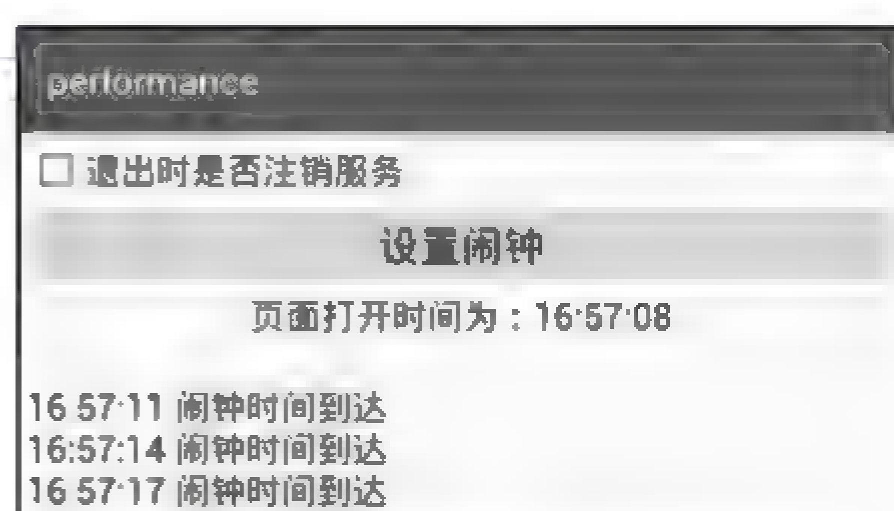


图 16-14　重新进入测试页面的情况

16.2.2　内存泄漏的预防

App 开发中的内存泄漏常见于以下 5 个场景：

（1）数据库查询操作后没有关闭游标 Cursor。

（2）适配器 Adapter 刷新数据时没有重用 convertView 对象。

（3）Bitmap 对象使用完毕没有调用 recycle 方法回收内存。

（4）Activity 引用了耗时对象，造成页面关闭时无法释放被引用的对象。

（5）给系统服务注册了监听任务，却没有及时注销。

要想避免出现内存泄漏，最好的办法是防患于未然。针对以上 5 个内存泄漏场景，相应的预防措施分别介绍如下：

1. 关闭游标

游标 Cursor 不止用于数据库 SQLite 查询记录，也可用于内容解析器 ContentResolver 查询内容数据，还可用于下载管理器 DownloadManager 查询下载进度。

若要预防游标产生的内存泄漏，则可在每次查询操作结束后调用 Cursor 对象的 close 方法关闭游标。

2. 重用适配

App 往列表视图 ListView 或网格视图 GridView 中填充数据都是通过适配器 BaseAdapter 的 getView 方法展示列表元素。列表元素较多时，系统只会加载屏幕上可见的元素，其他元素只有滑动到屏幕区域内才会即时加载并显示。当列表元素多次处于“展示→隐藏→展示→隐藏……”时，有必要重用每个元素的视图；如果不重用，那么每次展示可视元素都得重新分配视图对象，这便产生了内存泄露。

重用适配可先判断 convertView 对象，如果该对象为空，就为其分配视图对象，并调用 setTag 方法保存视图持有者；如果该对象非空，就调用 getTag 方法获取视图持有者。下面是重用列表元素的代码示例：

```
ViewHolder holder = null;
if (convertView == null) {
    holder = new ViewHolder();
    convertView = mInflater.inflate(R.layout.list_title, null);
    holder.tv_seq = (TextView) convertView.findViewById(R.id.tv_seq);
    holder.iv_title = (ImageView) convertView.findViewById(R.id.iv_title);
    convertView.setTag(holder);
} else {
    holder = (ViewHolder) convertView.getTag();
}
```

每次给 ListView 与 GridView 构造适配器都要加入上述重用代码，已经成了开发者的一大负担。所以 Android 在 5.0 之后推出了循环视图 RecyclerView，它的适配器自动实现视图持有者 ViewHolder，无须开发者进行重用判断的处理，算是一件善事。

3. 回收图像

Android 虽然定义了 Bitmap 类，但是读取图像数据的底层操作并非由 Java 代码完成。查看 SDK 源码，在 BitmapFactory 类中一路跟踪到 nativeDecodeStream 函数，发现它其实是一个 native 方法，也就是该方法来自于 JNI 接口。既然 Bitmap 的图像数据实际来自于 C/C++代码，那么确实得手工释放 C/C++的内存资源。查看 Bitmap 类的源码，它的回收方法 recycle 用到的 nativeRecycle 函数其实也是一个 native 方法，同样来自于 JNI 接口。



因此，若想避免图像操作引起的内存泄漏，可在 Bitmap 对象使用完毕后调用 recycle 方法。举一反三，只要一个资源是在 JNI 接口中分配的，一旦不再使用该资源，就得手工调用该资源对应的 JNI 回收接口。

4. 释放引用

编写 Handler 的处理函数时，Android Studio 提示 This Handler class should be static or leaks might occur，意思是这个类应该是一个静态类，否则可能发生内存泄漏。因为 Handler 对象经常处理异步任务，每当它调用 postDelayed 方法执行一个任务时，依据延迟间隔都得等待一段时间；倘若活动页面在此期间退出，就会导致异步任务持有的引用无法回收。由于 Runnable 通常持有 Activity 的引用，因此造成 Activity 资源都无法回收。

上面的描述可能不好理解，确实也不容易解释清楚，还是直接跳过烦琐的概念，讲讲如何解决该情况的内存泄漏问题。下面是预防这种内存泄漏的 3 个方法：

（1）如果异步任务是由 Handler 对象的 postDelayed 方法发起的，那么可用对应的 removeCallbacks 方法回收，把消息对象从消息队列移除就行了。

（2）按 Android 官方的推荐做法，可把 Handler 类改为静态类，同时 Handler 内部使用 WeakReference 关键字持有目标的引用。

之所以使用静态类，是因为静态类不持有目标的引用，不会影响内存自动回收机制。但是不持有目标的引用，Handler 内部就无法操作 Activity 上面的控件。为解决该问题，在构造 Handler 类时需要初始化目标的弱引用。不同于前面的强引用，弱引用相当于一个指针，指针指向的地址随时可以回收。这又带来一个新问题，即弱引用指向的对象可能是空的，所以 Handler 内部在使用目标活动前要先判断弱引用对象是否为空。

下面是弱引用的代码片段：

```
private static class MyHandler extends Handler {
    public static WeakReference<HandlerActivity> mActivity;
    public MyHandler(HandlerActivity activity) {
        mActivity = new WeakReference<HandlerActivity>(activity);
    }

    @Override
    public void handleMessage(Message msg) {
        HandlerActivity act = mActivity.get();
        if (act != null) {
            String desc = ProcessUtil.getRunningAppProcessInfo(act);
            act.tv_memory.setText(desc);
        }
    }
}
```

（3）把 Handler 对象作为 App 的全局变量，即把 Handler 对象作为自定义 Application 类的成员变量。

这样只要 App 在运行，该对象就一直存在。既然避免为 Handler 对象重复分配内存，也就间接避免了内存泄漏的可能。

5. 注销监听

App 的某些功能依赖于 Android 的系统服务，比如定位功能依赖于系统的定位管理器，定时功能依赖于系统的闹钟管理器。App 若想接收系统服务的消息，要么注册监听器，在回调方法中处理消息；要么注册广播接收器，在接收广播时处理消息。既然有注册操作，就存在对应的注销操作，不过如果不注意，就会忘记在代码中做注销处理。所以在进行页面编码时，千万要记得再检查一遍，确保 onDestroy 方法中已经包含相关的注销代码。

不同的系统服务拥有不同的注销方法，常见的系统服务注销方法见表 16-1。

表16-1 常见的系统服务注销方法

系统服务的管理器	注销操作说明	注销函数
AlarmManager	取消定时广播	cancel
ConnectivityManager	取消监听网络状态	unregisterNetworkCallback
DownloadManager	移除下载任务	remove
LocationManager	取消监听位置信息的变化	removeUpdates
LocationManager	取消监听定位状态的变化	removeGpsStatusListener
NotificationManager	取消通知	cancel
TelephonyManager	取消监听电话状态	使用 listen 方法注册一个空事件 PhoneStateListener.LISTEN_NONE
Vibrator	取消震动	cancel

16.3 线程池管理

在批量执行异步任务时，为了合理、有效地利用任务线程，需要引入线程池统一管理线程资源。就像数据库是对数据存储封装一样，线程池是对线程执行封装，总之都是为了提高系统的运行效率。本节先阐述单个线程存在的问题，然后依次说明普通线程池和定时器线程池的用法。

16.3.1 普通线程池

第 10 章介绍多线程时提到使用线程类 Thread 开启分线程，不过 Thread 只处理自身线程，缺乏多个线程之间的统一管理，会产生如下问题：

（1）无法控制线程的并发数，一旦同时启动多个线程，可能导致程序挂死。

（2）线程之间无法复用，每个线程都经历创建、启动、停止的生命周期，资源开销不小。

由于单线程管理存在诸多问题，因此异步任务工具 AsyncTask 给出了 executeOnExecutor

方法，允许开发者指定任务线程池。然而笔者当时已经指出，AsyncTask 自带的 THREAD_POOL_EXECUTOR 依然存在性能瓶颈。要想让线程池的处理性能达到最优，还得根据实际情况自定义线程池的具体参数。

Android 用到的是 Java 的线程池，由 Executors 类创建。系统已经封装好的线程池说明见表 16-2。

表16-2　已经封装好的线程池说明

线程池的创建方法	线程池类型	说明
newSingleThreadExecutor	ExecutorService	创建只有单个线程的线程池
newFixedThreadPool	ThreadPoolExecutor	创建线程数量固定的线程池
newCachedThreadPool	ThreadPoolExecutor	创建无个数限制的线程池
newSingleThreadScheduledExecutor	ScheduledThreadPoolExecutor	创建只有单个线程的定时器线程池
newScheduledThreadPool	ScheduledThreadPoolExecutor	创建线程数量固定的定时器线程池

当然，线程池中的线程数量最好由开发者分配，这时就要使用 ThreadPoolExecutor 的构造函数构建线程池对象。下面是构造函数的参数说明。

- int corePoolSize：线程池的最小线程个数。
- int maximumPoolSize：线程池的最大线程个数。
- long keepAliveTime：非核心线程在无任务时的等待时长。若超过该时间仍未分配任务，则该线程自动结束。
- TimeUnit unit：时间单位，时间单位的取值说明见表 16-3。

表16-3　时间单位的取值说明

TimeUnit 类的时间单位	说明
SECONDS	秒
MILLISECONDS	毫秒
MICROSECONDS	微秒

- BlockingQueue<Runnable> workQueue：设置等待队列。取值 new LinkedBlockingQueue<Runnable>()即可，默认表示等待队列无穷大，此时工作线程等于最小线程个数。当然也可在参数中指定等待队列的大小，此时工作线程数等于总任务数减去等待队列大小，工作线程数位于最小线程个数与最大线程个数之间。若计算得到的工作线程数小于最小线程个数，则工作线程数等于最小线程个数；若工作线程数大于最大线程个数，则系统扔出异常 java.util.concurrent.RejectedExecutionException，并不会自动让工作线程数等于最大线程个数。所以等待队列大小要么取默认值（不设置），要么设的尽可能大，否则一旦程序启动大量线程，就会异常报错。
- ThreadFactory threadFactory：一般使用默认值即可。

构建线程池对象后，还可在代码中随时调整参数，并执行任务管理操作。下面是 ThreadPoolExecutor 的常用方法说明。

- execute：向执行队列添加指定的任务。
- remove：从执行队列移除指定的任务。
- shutdown：关闭线程池。
- isTerminated：判断线程池是否关闭。
- setCorePoolSize：设置线程池的最小线程个数。
- setMaximumPoolSize：设置线程池的最大线程个数。
- setKeepAliveTime：设置非核心线程在无任务时的等待时长。
- getPoolSize：获取当前的线程个数。
- getActiveCount：获取当前的活动线程个数。

各种普通线程池的执行效果如图 16-15～图 16-18 所示。其中，图 16-15 所示为单线程线程池的结果界面，因为是单线程，所以每隔两秒打印一行日志；图 16-16 所示为多线程（4 个线程）线程池的结果界面，因为有 4 个线程，所以每秒打印 4 行日志；图 16-17 所示为无限制线程池的结果界面，因为不限制线程个数，所以一秒内就把所有日志打印出来了；图 16-18 所示为自定义线程（两个线程）线程池的结果界面，因为自定义了两个线程，所以每秒打印两行日志。

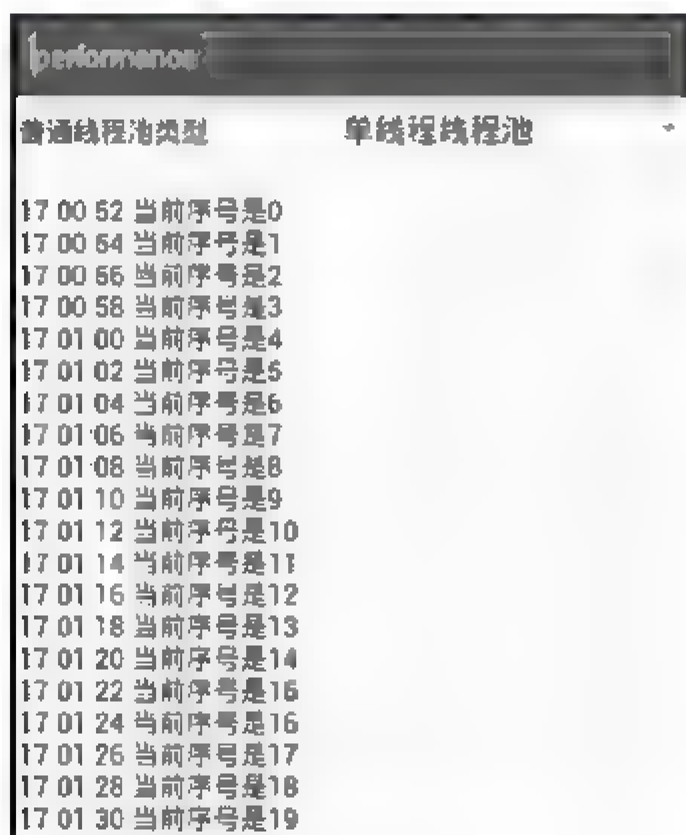


图 16-15　单线程线程池的日志



图 16-16　多线程线程池的日志

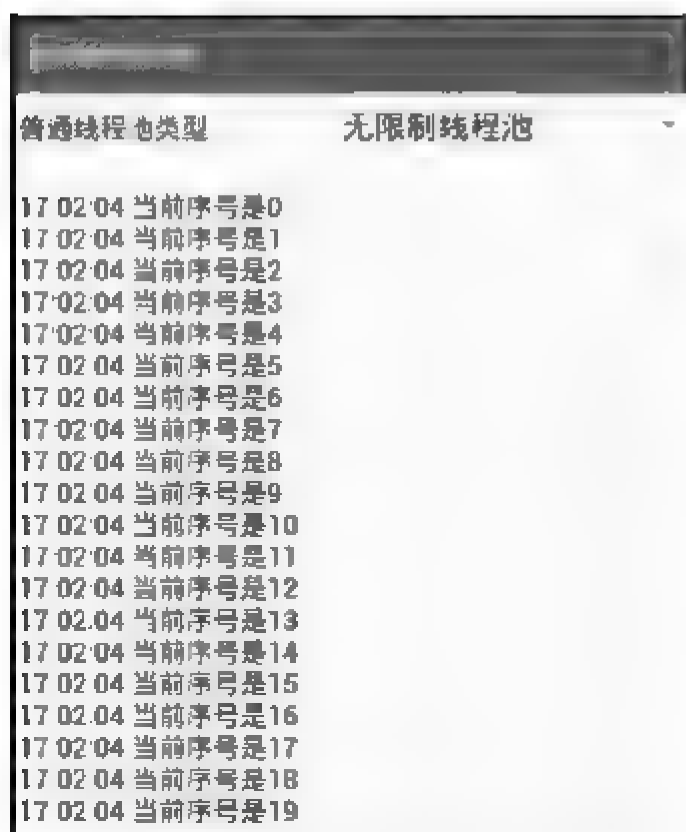


图 16-17　无限制线程池的日志

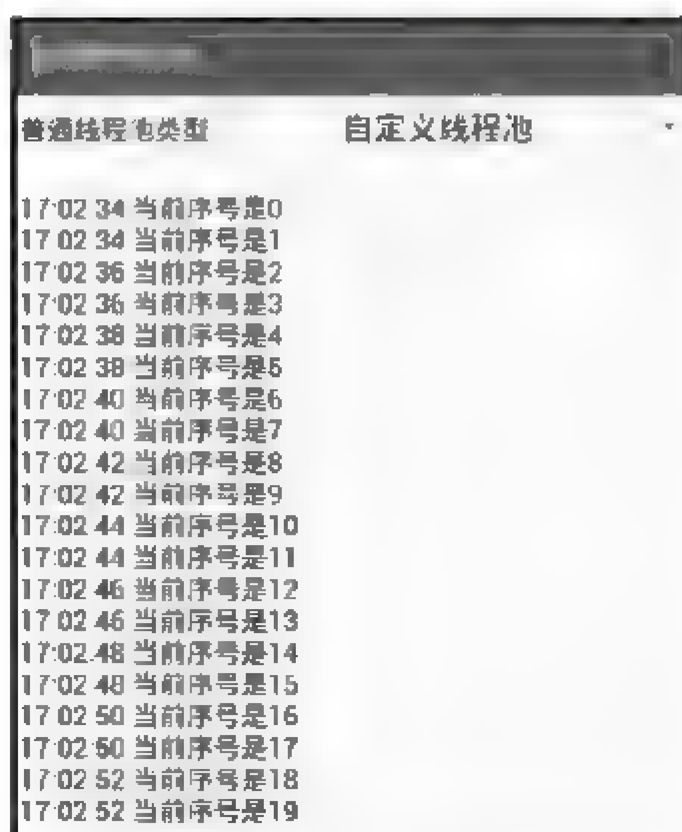


图 16-18　自定义线程池的日志

16.3.2　定时器线程池

前面的普通线程池是立即执行任务（如果有空余线程），但有时我们并不希望任务立即执行，而是延迟一段时间再执行，这样便用到了定时器线程池。

Android 同样提供了封装好的两个定时器线程池，即 newScheduledThreadPool 和 newSingleThreadScheduledExecutor，详细说明见表 16-2。当然现有的定时器线程池并不总能满足需求，还得由开发者自行定制。具体说来，就是使用 ScheduledThreadPoolExecutor 的构造函数构建定时器线程池对象。下面是构造函数的参数说明。

- int corePoolSize：线程池的最小线程个数。
- ThreadFactory threadFactory：一般使用默认值即可。ThreadFactory 是在线程池中使用的线程工厂接口，定义了一个 newThread 方法，该方法输入 Runnable 参数，返回 Thread 对象。虽然一般情况下使用默认的 DefaultThreadFactory 即可，但是在某些特定场合可以自己实现工厂类，用来跟踪线程的启动时间、结束时间，以及线程发生异常时的处理步骤。
- RejectedExecutionHandler handler：一般使用默认值即可。

定时器线程池 ScheduledExecutorService 继承了 ThreadPoolExecutor 的所有方法。下面是定时器线程池多出的几个定时器相关方法。

- schedule：延迟一段时间后启动任务。
- scheduleAtFixedRate：先延迟一段时间，然后间隔若干时间周期启动任务。
- scheduleWithFixedDelay：先延迟一段时间，然后固定延迟若干时间启动任务。

注意，scheduleAtFixedRate 和 scheduleWithFixedDelay 都是循环执行任务，区别在于前者的间隔时间从上个任务的开始时间起计算，后者的间隔时间从上个任务的结束时间起计算。

定时器线程池的执行效果如图 16-19 和图 16-20 所示。其中，图 16-19 所示为单线程的定时器线程池，每隔两秒打印一行日志；图 16-20 所示为多线程（3 个线程）的定时器线程池，因为有 3 个线程，所以每秒打印 3 行日志。

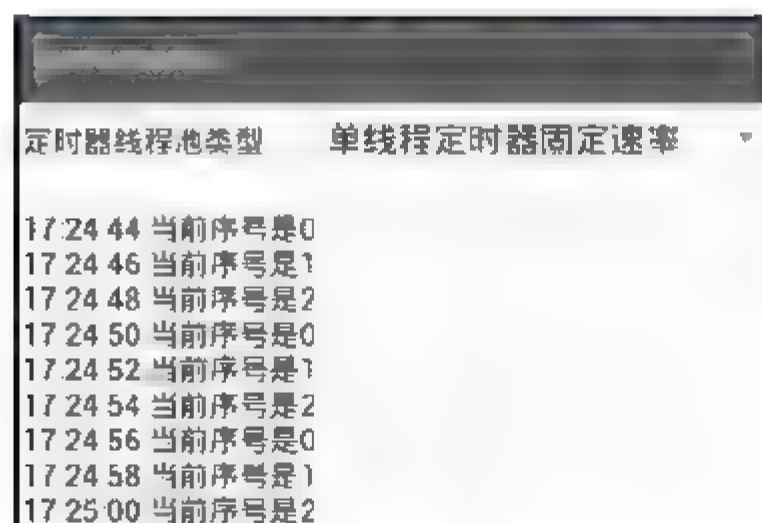


图 16-19　单线程定时器线程池的日志

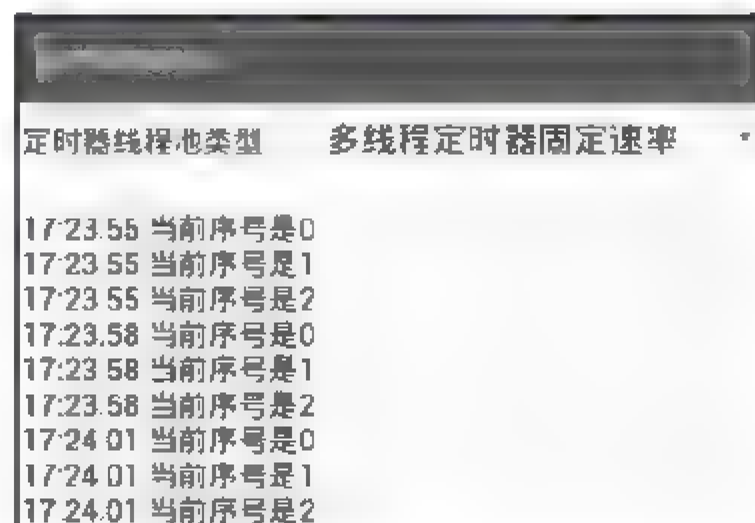


图 16-20　多线程定时器线程池的日志

16.4　省电模式

现在手机的电池容量越来越大，电量消耗的速度也越来越快，往往使用一两天手机就没

电了。电量跟流量是用户很关心的两个重要指标。一个App乱跑流量，很容易遭到用户抛弃；同样，一个App若是耗电大户，也难以逃脱被卸载的命运。所以App开发要注意适当省电，本节从电量检测与熄屏检测两方面论述如何开启自动省电模式。

16.4.1　检测当前电量

Android获取当前电量是通过监听广播实现的。具体地说，是监听电池的电量改变事件，即Intent.ACTION_BATTERY_CHANGED。因为接收该事件要求App处于活动状态，所以广播接收器不能在AndroidManifest.xml中注册，只能在代码中通过registerReceiver方法动态注册。注册完成即可监听电量变化广播，该广播携带的参数信息见表16-4。

表16-4　电量变化广播中携带的参数信息

BatteryManager 类的字段名称	字段获取方法	说明
EXTRA_SCALE	getIntExtra	电量刻度，通常是100
EXTRA_LEVEL	getIntExtra	当前电量
EXTRA_STATUS	getIntExtra	当前状态
BATTERY_STATUS_UNKNOWN	当前状态取值	未知
BATTERY_STATUS_CHARGING	当前状态取值	正在充电
BATTERY_STATUS_DISCHARGING	当前状态取值	正在断电
BATTERY_STATUS_NOT_CHARGING	当前状态取值	不在充电
BATTERY_STATUS_FULL	当前状态取值	电量充满
EXTRA_HEALTH	getIntExtra	健康程度
BATTERY_HEALTH_UNKNOWN	健康程度取值	未知
BATTERY_HEALTH_GOOD	健康程度取值	良好
BATTERY_HEALTH_OVERHEAT	健康程度取值	过热
BATTERY_HEALTH_DEAD	健康程度取值	坏了
BATTERY_HEALTH_OVER_VOLTAGE	健康程度取值	短路
BATTERY_HEALTH_UNSPECIFIED_FAILURE	健康程度取值	未知错误
BATTERY_HEALTH_COLD	健康程度取值	冷却
EXTRA_VOLTAGE	getIntExtra	当前电压
EXTRA_PLUGGED	getIntExtra	当前电源
0	当前电源取值	电池
BATTERY_PLUGGED_AC	当前电源取值	充电器
BATTERY_PLUGGED_USB	当前电源取值	USB
BATTERY_PLUGGED_WIRELESS	当前电源取值	无线
EXTRA_TECHNOLOGY	getStringExtra	当前技术，比如返回Li-ion表示锂电池
EXTRA_TEMPERATURE	getIntExtra	当前温度
EXTRA_PRESENT	getBooleanExtr	是否提供电池

检测当前电量的代码没什么技术含量，只是简单把电量变化广播中携带的参数信息打印出来。这里不再浪费篇幅，读者可参考本书下载资源中的相关实现。电量检测的效果如图 16-21 和图 16-22 所示。其中，图 16-21 所示为正在充电时的界面，图 16-22 所示为拔出充电器时的界面。



图 16-21　正在充电时的电量信息



图 16-22　没在充电时的电量信息

16.4.2　检测屏幕开关

大家很关心如何给 App 减负、省电，前人也做了不少总结工作，基本的判断原则是：越消耗资源的 App，耗电量就越大。具体到代码编写主要有以下省电措施：

（1）能用整型数计算就不用浮点数计算。
（2）能用 JSON 解析就不用 XML 解析。
（3）能用网络定位就不用 GPS 定位。
（4）尽量减少大文件的下载（如先压缩再下载、缓存已下载的文件）。
（5）用完系统资源要及时回收。
（6）能用线程处理就不用进程处理。
（7）多用缓存复用对象资源。如屏幕尺寸只需获取一次，之后可到缓存中读取。
（8）能用定时器广播就不用后台常驻服务。
（9）能用内存存储就不用文件存储。

上述省电措施虽然有效，但是比起耗电大户，还是小巫见大巫。在实际开发中，耗电大户其实是后台默默运行的 Service 服务。想想看，手机待机时，屏幕都不亮了，手机里面还有一些不知疲倦的 Service 在“愚公移山”，愚公也是要吃饭的呀。

既然如此，若想避免 App 在手机待机时仍在做无用功，可在熄屏时结束指定任务，在亮屏时再开始指定任务。其中，熄屏事件监听的是系统广播 Intent.ACTION_SCREEN_ON，亮屏事件监听的是系统广播 Intent.ACTION_SCREEN_OFF。

在具体的编码中，监听这两个屏幕事件需要注意以下 3 点：

（1）熄屏事件和亮屏事件必须在代码中动态注册。如果在 AndroidManifest.xml 中静态注册，就不起任何作用。



（2）在熄屏时，系统先暂停所有活动页面，然后才关闭屏幕；同样，在亮屏时，系统先点亮屏幕，然后才恢复活动页面。这两个事件不能在 Activity 代码中注册和注销，只能在自定义 Application 类的 onCreate 方法中注册广播接收器。

（3）活动页面要想得知屏幕开关的事件信息，必须通过自定义的 Application 类间接获取。

检测屏幕开关事件的代码如下：

```
public class MainApplication extends Application {
    private static final String TAG = "MainApplication";
    private static MainApplication mApp;
    private LockScreenReceiver mReceiver;
    private String mChange = "";

    public static MainApplication getInstance() {
        return mApp;
    }

    public String getChangeDesc() {
        return mApp.mChange;
    }

    public void setChangeDesc(String change) {
        mApp.mChange = mApp.mChange + change;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        mApp = this;
        mReceiver = new LockScreenReceiver();
        IntentFilter filter = new IntentFilter();
        filter.addAction(Intent.ACTION_SCREEN_ON);
        filter.addAction(Intent.ACTION_SCREEN_OFF);
        filter.addAction(Intent.ACTION_USER_PRESENT);
        registerReceiver(mReceiver, filter);
    }

    private class LockScreenReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            if (intent != null) {
                String change = "";
                change = String.format("%s\n%s ： 收到广播 ： %s", change,
                        Utils.getNowTime(), intent.getAction());
                if (intent.getAction().equals(Intent.ACTION_SCREEN_ON)) {
                    change = String.format("%s\n 这是屏幕点亮事件，可在此开启日常操作",
```

```
change);
                    } else if (intent.getAction().equals(Intent.ACTION_SCREEN_OFF)) {
                        change = String.format("%s\n 这是屏幕关闭事件，可在此暂停耗电操作",
change);
                    } else if (intent.getAction()
                            .equals(Intent.ACTION_USER_PRESENT)) {
                        change = String.format("%s\n 这是用户解锁事件", change);
                    }
                    Log.d(TAG, change);
                    MainApplication.getInstance().setChangeDesc(change);
                }
            }
        }
    }
```

屏幕检测的效果如图 16-23 所示，能够正确检测到熄屏事件和亮屏事件，才可执行后台服务的停止与启动操作。



图 16-23　测试应用监测到了熄屏事件和亮屏事件

16.5　实战项目：图片缓存框架

性能优化说来说去，归根到底是用最少的资源换取最高的效率，也就是看哪个性价比最高。在性能优化的诸多措施中，性价比最高的当数图片缓存，App 要想既好看又丰富，都是靠大量图片堆砌出来的。与其纠结 HTTP 交互文本采用 JSON 格式好还是采用 XML 格式好，还不如好好研究图片缓存技术，一张图片的运算量远远超过一段文本。况且图片缓存不但可以加快运行速度，而且能节省流量，还能省电，从而极大改善用户体验。本章以“图片缓存框架”实战项目作为结尾。

16.5.1　设计思路

第 4 章的实战项目“购物车”中已经出现了图片缓存的雏形，当时的图片缓存只有两级，即“全局内存”→“SD 卡文件”。实际开发中的图片缓存至少为 3 级，即“内存”→“SD 卡”→“网络”。正常情况下，App 先到内存中寻找图片，如果找到，就直接显示内存中的图



片。如果在内存中没找到图片，再到SD卡寻找。如果在SD卡找到图片，就读取SD卡图片并显示。如果在SD卡也没找到，就得根据URL去网络下载图片，下载成功后再显示图片。经过3级缓存查找，即使网速很慢甚至断网，App也能迅速加载大部分图片，使得用户的浏览操作基本不受影响。

当然，图片缓存技术主要在后台实现，普通用户不容易感觉到它的存在。不过只要稍加注意，你也能发现界面上采用图片缓存的端倪。如果一张图片以灰度动画逐渐显示时，很可能就是通过缓存技术加载的。图16-24和图16-25分别展示了图片加载的开始与完成界面，图16-24中的玫瑰花尚且朦朦胧胧、若隐若现，提示用户当前图片正在加载；加载完毕后，整张玫瑰花图片清晰地显示出来，如图16-25所示。



图16-24 玫瑰花图片正在加载



图16-25 玫瑰花图片加载完成

在技术上，灰度动画开始时，整张图片已经下载完成。之所以加入动画的渐变效果，是为了留出缓冲的过程，让用户不至于觉得图片一闪一闪很突兀。接下来看图片缓存框架用到了哪些App技术。

（1）图像视图ImageView：无论多么高深的框架，都要打好基础。

（2）灰度动画AlphaAnimation：通过渐变效果展示图片加载的过程，用到了灰度动画。

（3）图片的基本加工：有时为了减少资源占用，仅需展示缩略图，用户有需要再显示大图。

（4）内存的读写：多张图片保存在缓存队列中，要求有合适的数据结构进行管理。

（5）SD卡的文件读写：从网络下载的图片先保存在SD卡，再依情况决定是否加载进内存。

（6）HTTP访问：从网络获取图片，可直接从HTTP地址读取图片数据。

（7）多线程：网络访问请求，需要开启分线程处理，并操作Handler对象。

（8）线程池：页面同时请求多张图片，需要线程池统一管理图片下载的各线程资源。

（9）内存泄漏：频繁操作Handler对象，要及时释放该对象的引用。另外，对Bitmap对象也要注意加以回收。

上面一口气列举了这么多知识点，原来图片缓存是一个综合技术活，可算是集Android技术大全了。只要理解图片缓存的算法，并加以实践将其做好，就差不多可以掌握半部App的开发。

16.5.2 小知识：LRU 缓存策略

读者也许还记得大学里操作系统课程中的页面置换算法，说的是操作系统发现要访问的数据在内存中找不到，只好把内存中很久没用的页面踢出去，以便给本次访问的数据让出存储空间。图片缓存的排队算法类似页面置换算法，常见的主要有两种：FIFO 先进先出算法和 LRU 最久未使用算法。FIFO 算法比较容易实现，只要把数据按时间先后顺序排队，要淘汰老旧数据时，只需把队列最前端的数据移除即可。因为图片缓存的 FIFO 算法需要对队列两端进行操作，从队列顶端移除淘汰的图像，并把新增的图像加到队列末端，所以该算法的缓存结构可采用双端队列 LinkedList。

麻烦的是 LRU 算法，虽然该算法在实际开发中用得最多，缓存效果也最好，但 Java 却没提供该算法对应的数据结构。幸好 Android 在设计之初已经考虑了该问题，提供了 LruCache 缓存工具，方便开发者实现 LRU 算法的相关缓存业务。LruCache 内部集成了缓存数据的插入时间判断，无论缓存内部是否已经存在某键值，新插入的键数据总是位于缓存队列尾部，如此开发者不必关心具体的排队淘汰逻辑，只需进行 App 的业务处理就好。

下面是 LruCache 的常用方法说明。

- 构造函数：初始化指定大小的缓存队列。
- resize：变更缓存队列的大小。
- put：往缓存队列插入数据。
- get：从缓存队列获取数据。
- remove：把指定数据移出缓存队列。
- evictAll：清空缓存队列。
- size：获得已使用的缓存队列大小。
- maxSize：获得缓存队列的总大小。
- snapshot：获得缓存队列的快照，即获取缓存队列当前的映射表。

可以看出，LruCache 的用法与 Java 的容器类差不多，但有两个不同点值得注意一下：

（1）LruCache 未提供 contains 函数用于判断某键值是否存在，只能调用 get 函数间接判断。即检查 get 函数的返回值，如果返回 null 就表示缓存中不存在该键值。

（2）LruCache 不能直接进行遍历操作，只能调用 snapshot 函数获得当前快照，再遍历快照中的映射表 Map。

接下来通过实际代码加深对 LruCache 的理解，示例代码如下：

```
public class LruCacheActivity extends AppCompatActivity implements OnClickListener {
    private TextView tv_lru_cache;
    private LruCache<String, String> mLanguageLru;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_lru_cache);
```

```
        tv_lru_cache = (TextView) findViewById(R.id.tv_lru_cache);
        findViewById(R.id.btn_android).setOnClickListener(this);
        findViewById(R.id.btn_ios).setOnClickListener(this);
        findViewById(R.id.btn_java).setOnClickListener(this);
        findViewById(R.id.btn_cpp).setOnClickListener(this);
        findViewById(R.id.btn_python).setOnClickListener(this);
        findViewById(R.id.btn_net).setOnClickListener(this);
        findViewById(R.id.btn_php).setOnClickListener(this);
        findViewById(R.id.btn_perl).setOnClickListener(this);
        mLanguageLru = new LruCache<String, String>(5);
    }

    @Override
    public void onClick(View v) {
        String language = ((Button) v).getText().toString();
        mLanguageLru.put(language, Utils.getNowTime());
        printCache();
    }

    private void printCache() {
        String desc = "";
        Map<String, String> cache = mLanguageLru.snapshot();
        for (Map.Entry<String, String> item : cache.entrySet()) {
            desc = String.format("%s%s 最后一次更新时间为%s\n",
                    desc, item.getKey(), item.getValue());
        }
        tv_lru_cache.setText(desc);
    }
}
```

上述代码运行后的效果如图 16-26～图 16-29 所示。其中，图 16-26 所示为 LRU 缓存在某个时刻的快照，此时 Android 位于队列顶端；然后点击 ANDROID 按钮，表示现在已访问 Android，于是 Android 从队列顶端移到了队列底部，并且插入时间也被更新了，此时队列顶端的数据变成了 iOS，如图 16-27 所示。



图 16-26　LRU 缓存队列里的初始数据



图 16-27　点击 Android 后的缓存队列

接着点击 PHP 按钮，表示访问 PHP 语言，于是 PHP 被插入到缓存队列底部，同时顶端

的 iOS 被移出队列，此时队列顶端的数据变成了 JAVA，如图 16-28 所示。然后点击 JAVA 按钮，表示访问 Java 语言，于是 Java 从队列顶端移到了队列底部，而且更新了插入时间，此时队列顶端的数据变成了 C/C++，如图 16-29 所示。



图 16-28　点击 PHP 后的 LRU 缓存队列



图 16-29　点击 Java 后的 LRU 缓存队列

16.5.3　代码示例

编码与测试方面需要注意以下 3 点：

（1）AndroidManifest.xml 注意声明相关权限，举例如下：

```
<!-- 上网 -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- SD 卡 -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
```

（2）除了普通控件的图片缓存，还要实现列表视图里的图片缓存。因为 ListView 只会加载当前屏幕上可见的列表元素，通过不断上拉与下拉 ListView，可以观察图片缓存是否正常工作，以及是否发生内存泄漏的情况。

（3）使用真机对联网与断网两种情况分别进行测试。

最后是图片缓存框架的演示时间，在加载图片前，通常在原图位置放一张占位图片，如图 16-30 所示。如果图片加载失败，就在原图位置显示出错图片，提示用户原图加载失败，如图 16-31 所示。



图 16-30　加载前先显示占位图片

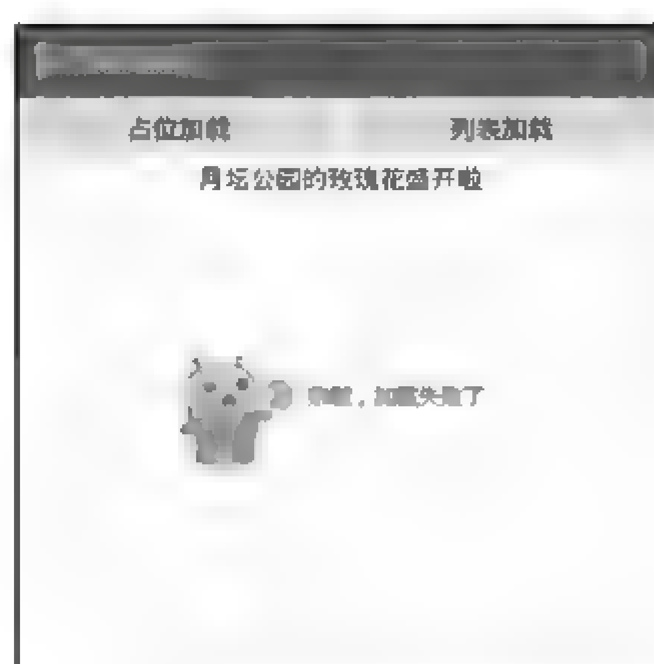


图 16-31　加载失败显示出错图片

在联网的情况下，只要图片地址准确，图片缓存框架便能正常工作。一旦从“内存”→“SD 卡”→“网络”3 级缓存中取到图片，便可通过渐变动画在页面展示图片。如图 16-32 所示，此时动画开始，图片正逐步变亮；等到动画结束，图片揭开面纱，完全呈现出来，如图 16-33 所示。

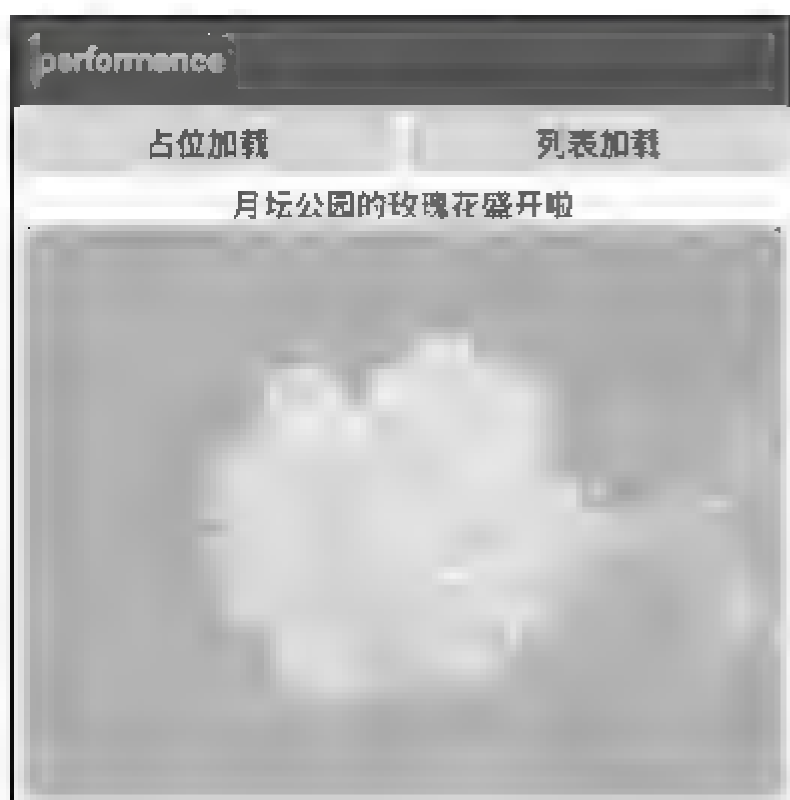


图 16-32 渐变动画正在播放



图 16-33 渐变动画结束播放

再来看图片缓存在列表视图中是如何工作的。如图 16-34 所示，一打开 ListView 图片列表页面，处于屏幕可视区域的前两张图片就开始加载；待加载完毕，这两张图片清晰展现开来，如图 16-35 所示。

然后把页面拉到底部，原本处于不可见区域的最后两项开始加载，图片逐渐变亮，如图 16-36 所示；最终这两张图片完整无缺地显示出来，如图 16-37 所示。

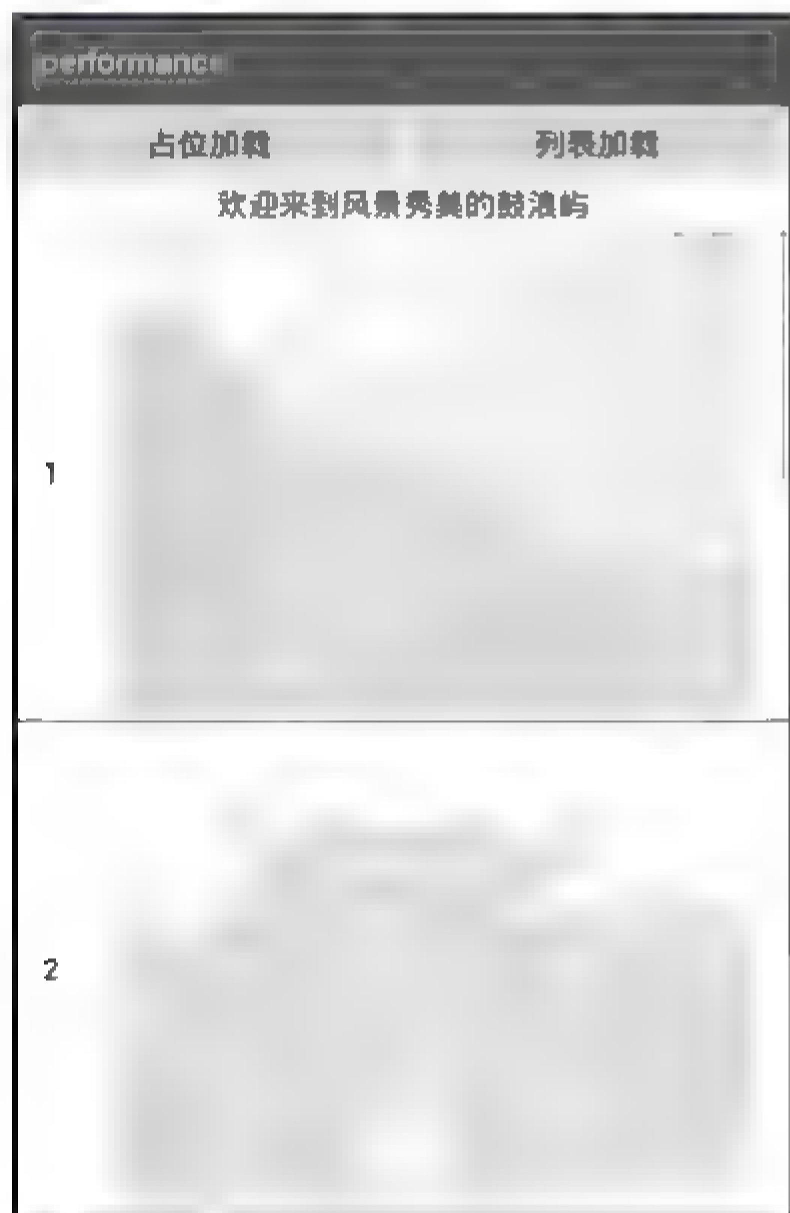


图 16-34 列表视图头部图片开始加载

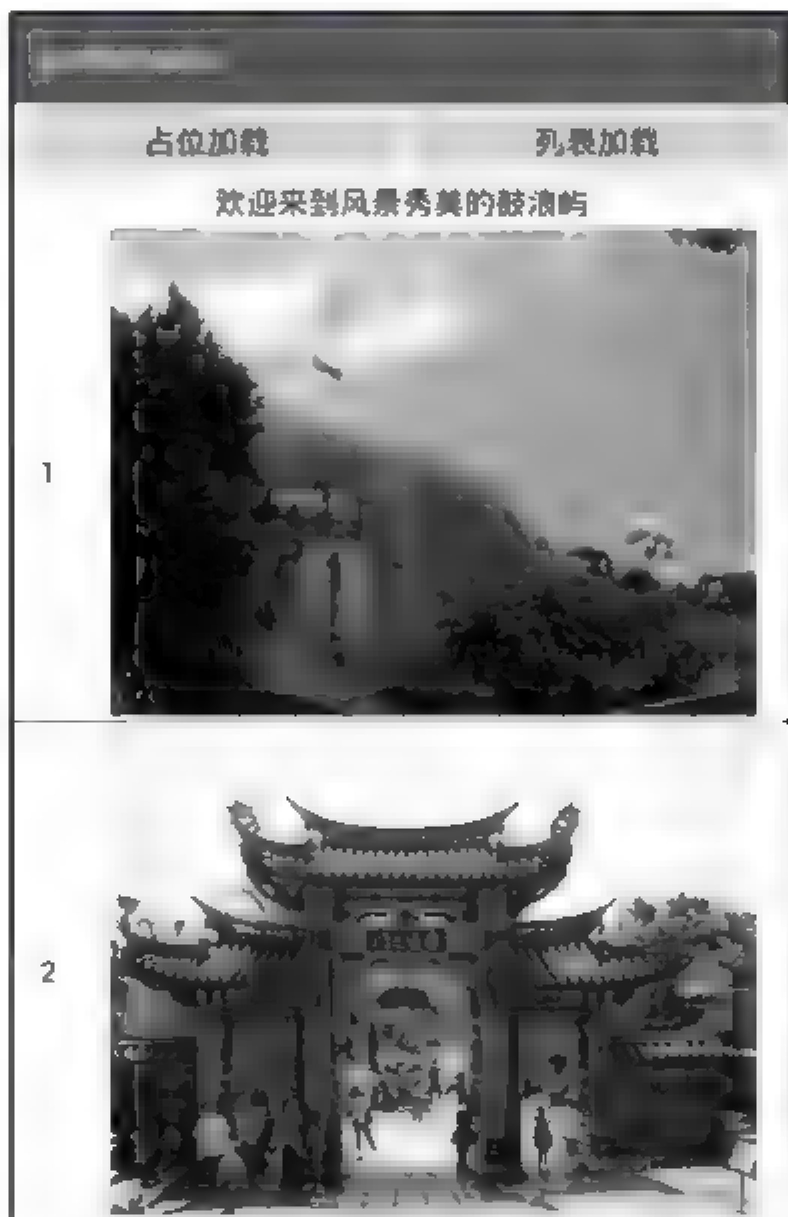


图 16-35 列表视图头部图片结束加载

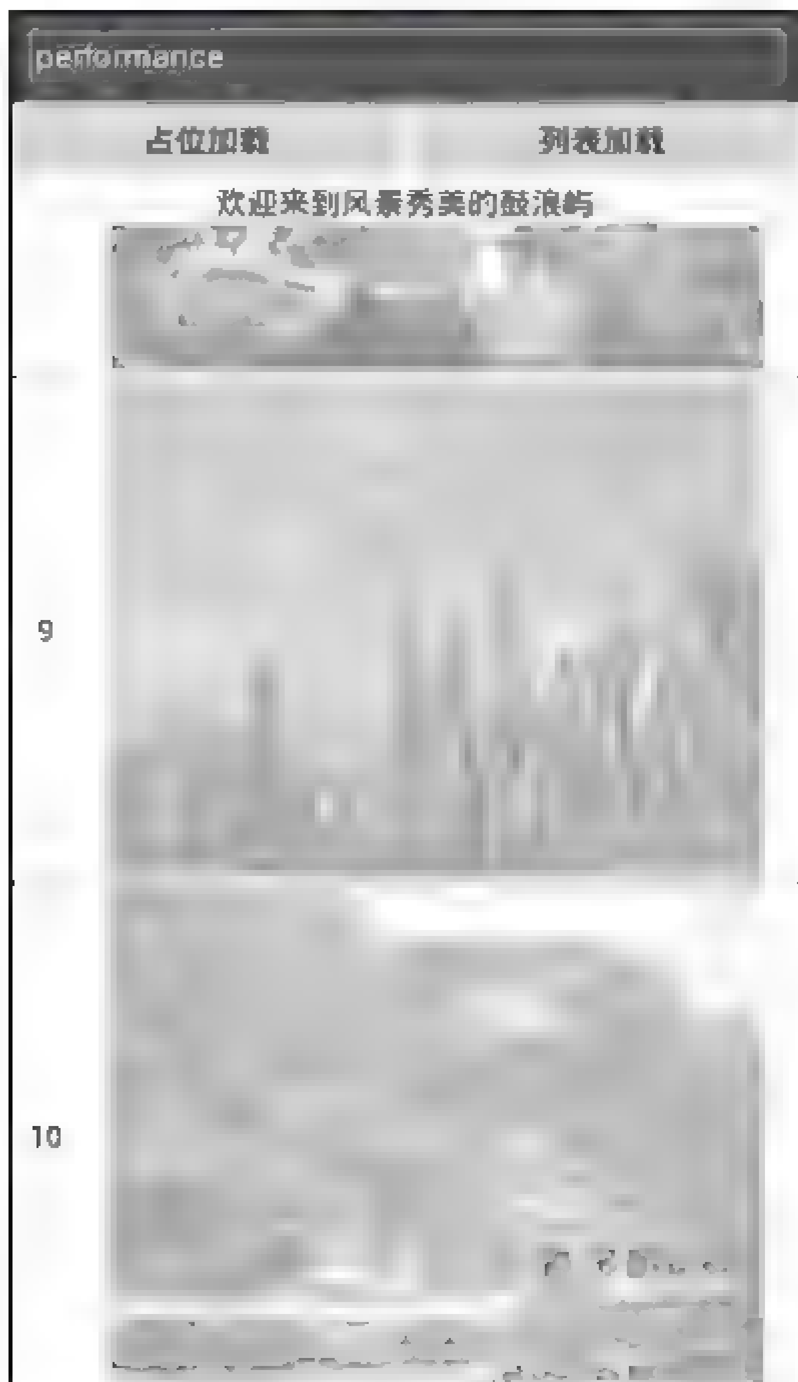


图 16-36　列表视图底部图片开始加载

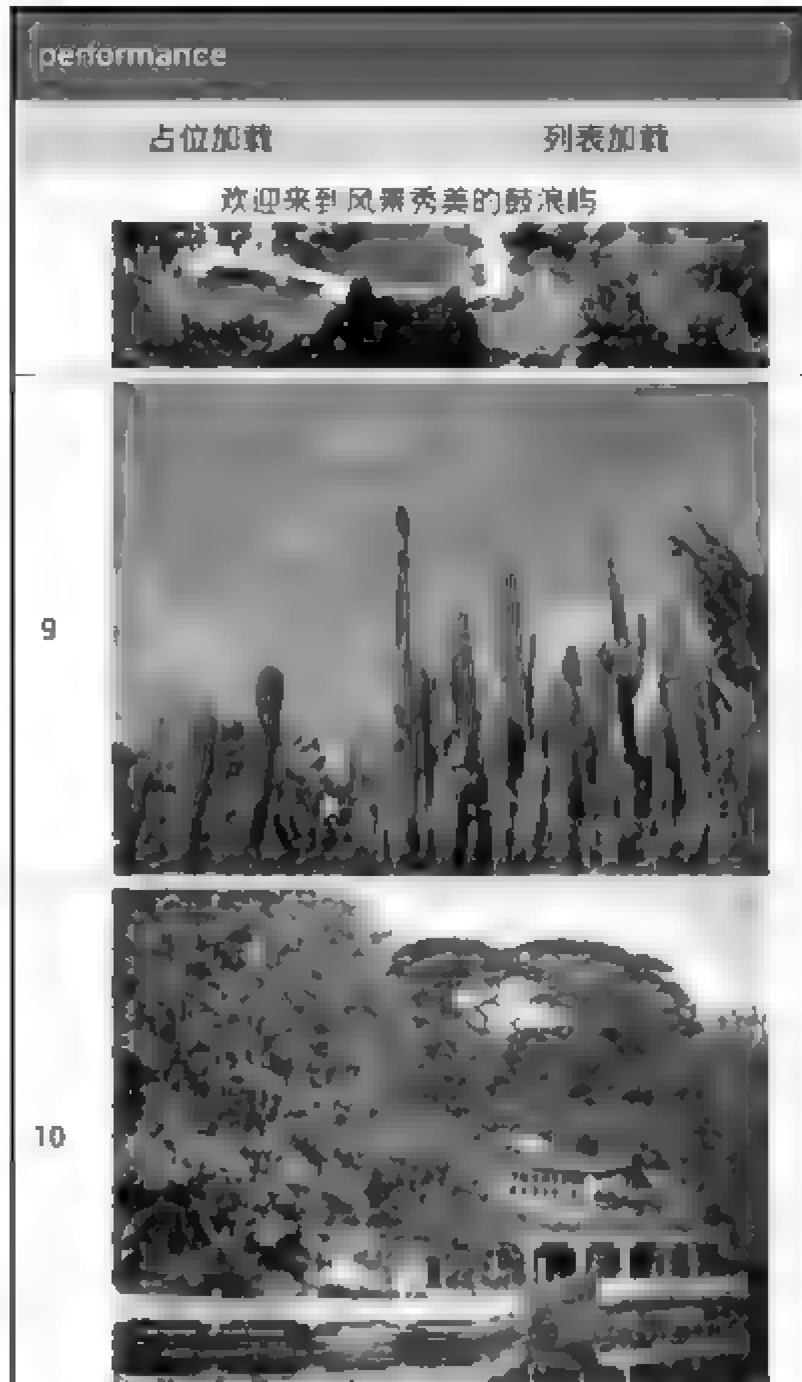


图 16-37　列表视图底部图片结束加载

看起来是不是很神奇？图片缓存框架的用法很简单，先设置好各项缓存的处理参数，然后调用 show 方法即可。下面是该缓存框架的调用代码示例：

```
ImageCacheConfig config = new ImageCacheConfig.Builder()
        .setBeginImage(R.drawable.load_default)     //设置初始图片
        .setErrorImage(R.drawable.load_error)       //设置出错图片
        .setCacheStyle(ImageCacheConfig.LRU)        //设置 LRU 缓存算法
        .setFadeInterval(2000).build();   //设置渐变动画的持续时间
//初始化缓存设置，并把地址为 file 的图片显示到控件 iv_cache 上
mCache.initConfig(config).show(file, iv_cache);
```

图片缓存框架的核心处理代码如下：

```
public class ImageCache {
    private final static String TAG = "ImageCache";
    //内存中的图片缓存
    private HashMap<String, Bitmap> mImageMap = new HashMap<String, Bitmap>();
    //uri 与视图控件的映射关系
    private HashMap<String, ImageView> mViewMap = new HashMap<String, ImageView>();
    //缓存队列，采用 FIFO 先进先出策略，需操作队列首尾两端，故采用双端队列
    private LinkedList<String> mFifoList = new LinkedList<String>();
    //缓存队列，采用 LRU 近期最少使用策略，Android 专门提供了 LruCache 实现该算法
    private LruCache<String, Bitmap> mImageLru;
```

```
    private ImageCacheConfig mConfig;
    private String mDir = "";
    private ThreadPoolExecutor mPool;
    private static Handler mMyHandler;
    private static ImageCache mCache = null;
    private static Context mContext;

    public static ImageCache getInstance(Context context) {
        if (mCache == null) {
            mCache = new ImageCache();
            mCache.mContext = context;
        }
        return mCache;
    }

    public ImageCache initConfig(ImageCacheConfig config) {
        mCache.mConfig = config;
        mCache.mDir = mCache.mConfig.mDir;
        if (mCache.mDir==null || mCache.mDir.length()<=0) {
            mCache.mDir = Environment.getExternalStorageDirectory() + "/image_cache";
        }
        Log.d(TAG, "mDir="+mCache.mDir);
        File dir = new File(mCache.mDir);
        if (dir.exists() != true) {
            dir.mkdirs();   // 若目录不存在，则先创建新目录
        }
        mCache.mPool = (ThreadPoolExecutor) Executors.newFixedThreadPool
(mCache.mConfig.mThreadCount);
        mCache.mMyHandler = new MyHandler((Activity)mCache.mContext);
        if (mCache.mConfig.mCacheStyle == ImageCacheConfig.LRU) {
            mImageLru = new LruCache(mCache.mConfig.mMemoryFileCount);
        }
        return mCache;
    }

    public void show(String uri, ImageView iv) {
        if (mConfig.mBeginImage != 0) {
            iv.setImageResource(mConfig.mBeginImage);
        }
        mViewMap.put(uri, iv);
        if (checkExist(uri) == true) {
            mCache.render(uri, getBitmap(uri));
        } else {
```

```
                String path = getFilePath(uri);
                if ((new File(path)).exists() == true) {
                    Bitmap bitmap = ImageUtil.openBitmap(path);
                    if (bitmap != null) {
                        mCache.render(uri, bitmap);
                    } else {
                        mPool.execute(new MyRunnable(uri));
                    }
                } else {
                    mPool.execute(new MyRunnable(uri));
                }
            }
        }

        private boolean checkExist(String uri) {
            if (mCache.mConfig.mCacheStyle == ImageCacheConfig.LRU) {
                return (mImageLru.get(uri)==null)?false:true;
            } else {
                return mImageMap.containsKey(uri);
            }
        }

        private Bitmap getBitmap(String uri) {
            if (mCache.mConfig.mCacheStyle == ImageCacheConfig.LRU) {
                return mImageLru.get(uri);
            } else {
                return mImageMap.get(uri);
            }
        }

        private String getFilePath(String uri) {
            String file_path = String.format("%s/%d.jpg", mDir, uri.hashCode());
            return file_path;
        }

        private static class MyHandler extends Handler {
            public static WeakReference<Activity> mActivity;
            public MyHandler(Activity activity) {
                mActivity = new WeakReference<Activity>(activity);
            }

            @Override
            public void handleMessage(Message msg) {
```

```
                Activity act = mActivity.get();
                if (act != null) {
                    ImageData data = (ImageData) (msg.obj);
                    if (data!=null && data.bitmap!=null) {
                        mCache.render(data.uri, data.bitmap);
                    } else {
                        mCache.showError(data.uri);
                    }
                }
            }
        }

        private class MyRunnable implements Runnable {
            private String mUri;
            public MyRunnable(String uri) {
                mUri = uri;
            }

            @Override
            public void run() {
                Activity act = MyHandler.mActivity.get();
                if (act != null) {
                    Bitmap bitmap = ImageHttp.getImage(mUri);
                    if (bitmap != null) {
                        if (mConfig.mSize != null) {
                            bitmap = Bitmap.createScaledBitmap(bitmap, mConfig.mSize.x,
mConfig.mSize.y, false);
                        }
                        ImageUtil.saveBitmap(getFilePath(mUri), bitmap);
                    }
                    ImageData data = new ImageData(mUri, bitmap);
                    Message msg = mMyHandler.obtainMessage();
                    msg.obj = data;
                    mMyHandler.sendMessage(msg);
                }
            }
        };

        private void render(String uri, Bitmap bitmap) {
            ImageView iv = mViewMap.get(uri);
            if (mConfig.mFadeInterval <= 0) {
                iv.setImageBitmap(bitmap);
            } else {
```

```
            //内存中已有图片的就直接显示，不再展示淡入淡出动画
            if (checkExist(uri) == true) {
                iv.setImageBitmap(bitmap);
            } else {
                iv.setAlpha(0.0f);
                AlphaAnimation alphaAnimation = new AlphaAnimation(0.0f, 1.0f);
                alphaAnimation.setDuration(mConfig.mFadeInterval);
                alphaAnimation.setFillAfter(true);
                iv.setImageBitmap(bitmap);
                iv.setAlpha(1.0f);
                iv.setAnimation(alphaAnimation);
                alphaAnimation.start();
                mCache.refreshList(uri, bitmap);
            }
        }
    }

    private synchronized void refreshList(String uri, Bitmap bitmap) {
        if (mCache.mConfig.mCacheStyle == ImageCacheConfig.LRU) {
            mImageLru.put(uri, bitmap);
        } else {
            if (mFifoList.size() >= mConfig.mMemoryFileCount) {
                String out_uri = mFifoList.pollFirst();
                mImageMap.remove(out_uri);
            }
            mImageMap.put(uri, bitmap);
            mFifoList.addLast(uri);
        }
    }

    private void showError(String uri) {
        ImageView iv = mViewMap.get(uri);
        if (mConfig.mErrorImage != 0) {
            iv.setImageResource(mConfig.mErrorImage);
        }
    }

    public void clear() {
        for (Map.Entry<String, Bitmap> item_map : mImageMap.entrySet()) {
            Bitmap bitmap = item_map.getValue();
            bitmap.recycle();
        }
        if (mImageLru != null) {
```

```
                mImageLru.evictAll();
            }
            mCache = null;
        }
    }
```

16.6 小　结

本章主要介绍了 App 开发用到的常见性能优化技术，包括布局文件优化（减少重复布局、自适应调整布局、自定义窗口主题）、内存泄漏处理（内存泄漏的检测、内存泄漏的预防）、线程池管理（普通线程池、定时器线程池）、省电模式（检测当前电量、检测屏幕开关）。最后设计了一个实战项目“图片缓存框架”，在该项目的 App 编码中采用了本书讲述的与存储和多线程有关的主要技术。另外，介绍了 LRU 缓存策略的原理与用法。

通过本章的学习，读者应该能够掌握以下 4 种开发技能：

（1）学会使用布局文件优化技术统一界面风格。
（2）学会检测内存泄漏的情况，并采取相应的预防措施。
（3）学会有效使用和管理线程池。
（4）学会图片缓存框架的基本原理和具体实现。